INVITED PAPER *Special Section on Signal Processing*

Fast Local Algorithms for Large Scale Nonnegative Matrix and Tensor Factorizations

Andrzej CICHOCKI[†a)], ***Member and*** **Anh-Huy PHAN**[††b)], ***Nonmember***

SUMMARY Nonnegative matrix factorization (NMF) and its extensions such as Nonnegative Tensor Factorization (NTF) have become prominent techniques for blind sources separation (BSS), analysis of image databases, data mining and other information retrieval and clustering applications. In this paper we propose a family of efficient algorithms for NMF/NTF, as well as sparse nonnegative coding and representation, that has many potential applications in computational neuroscience, multisensory processing, compressed sensing and multidimensional data analysis. We have developed a class of optimized local algorithms which are referred to as Hierarchical Alternating Least Squares (HALS) algorithms. For these purposes, we have performed sequential constrained minimization on a set of squared Euclidean distances. We then extend this approach to robust cost functions using the Alpha and Beta divergences and derive flexible update rules. Our algorithms are locally stable and work well for NMF-based blind source separation (BSS) not only for the over-determined case but also for an under-determined (over-complete) case (i.e., for a system which has less sensors than sources) if data are sufficiently sparse. The NMF learning rules are extended and generalized for *N*-th order nonnegative tensor factorization (NTF). Moreover, these algorithms can be tuned to different noise statistics by adjusting a single parameter. Extensive experimental results confirm the accuracy and computational performance of the developed algorithms, especially, with usage of multi-layer hierarchical NMF approach [3].

key words: *Nonnegative matrix factorization (NMF), nonnegative tensor factorizations (NTF), nonnegative PARAFAC, model reduction, feature extraction, compression, denoising, multiplicative local learning (adaptive) algorithms, Alpha and Beta divergences.*

1. Introduction

Recent years have seen a surge of interest in nonnegative and sparse matrix and tensor factorization - decompositions which provide physically meaningful latent (hidden) components or features. Nonnegative Matrix Factorization (NMF) and its extension Nonnegative Tensor Factorization (NTF) - multidimensional models with nonnegativity constraints - have been recently proposed as sparse and efficient representations of signals, images and in general natural signals/data. From signal processing point of view and data analysis, NMF/NTF are very attractive because they take into account spatial and temporal correlations between variables and usually provide sparse common factors or hidden (latent) nonnegative components with physical or physiological meaning and interpretations [1]–[5].

In fact, NMF and NTF are emerging techniques for data mining, dimensionality reduction, pattern recognition, object detection, classification, gene clustering, sparse nonnegative representation and coding, and blind source separation (BSS) [5]–[14]. For example, NMF/NTF have already found a wide spectrum of applications in positron emission tomography (PET), spectroscopy, chemometrics and environmental science where the matrices have clear physical meanings and some normalization or constraints are imposed on them [12], [13], [15].

This paper introduces several alternative approaches and improved local learning rules (in the sense that vectors and rows of matrices are processed sequentially one by one) for solving nonnegative matrix and tensor factorizations problems. Generally, tensors (i.e., multi-way arrays) are denoted by underlined capital boldface letter, e.g., $\underline{\boldsymbol{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$. The order of a tensor is the number of modes, also known as ways or dimensions. In contrast, matrices are denoted by boldface capital letters, e.g., $\boldsymbol{Y}$; vectors are denoted by boldface lowercase letters, e.g., columns of the matrix $\boldsymbol{A}$ by $\boldsymbol{a}_j$ and scalars are denoted by lowercase letters, e.g., a_{ij}. The i-th entry of a vector $\boldsymbol{a}$ is denoted by a_i, and (i, j) element of a matrix $\boldsymbol{A}$ by a_{ij}. Analogously, element (i, k, q) of a third-order tensor $\underline{\boldsymbol{Y}} \in \mathbb{R}^{I \times K \times Q}$ by y_{ikq}. Indices typically range from 1 to their capital version, e.g., $i = 1, 2, \ldots, I$; $k = 1, 2, \ldots, K$; $q = 1, 2, \ldots, Q$. Throughout this paper, standard notations and basic tensor operations are used as indicated in Table 1.

Table 1 Basic tensor operations and notations [16]

$\circ$	outer product	$\times_n$	$n-mode$ product of tensor and matrix
$\odot$	Khatri-Rao product	$\overline{\times}_n$	$n-mode$ product of tensor and vector
$\otimes$	Kronecker product	$\boldsymbol{Y}_{(n)}$	$n-mode$ matricized version of $\underline{\boldsymbol{Y}}$
$\circledast$	Hadamard product	$\boldsymbol{U}^{\odot}$	$\boldsymbol{U}^{(N)} \odot \boldsymbol{U}^{(N-1)} \odot \cdots \odot \boldsymbol{U}^{(1)}$
$\oslash$	element-wise division	$\boldsymbol{U}^{\odot -n}$	$\boldsymbol{U}^{(N)} \odot \cdots \odot \boldsymbol{U}^{(n+1)} \odot \boldsymbol{U}^{(n-1)} \odot \cdots \odot \boldsymbol{U}^{(1)}$
$[\boldsymbol{U}]_j$	jth column vector of $[\boldsymbol{U}]$	$\boldsymbol{U}^{\circledast}$	$\boldsymbol{U}^{(N)} \circledast \boldsymbol{U}^{(N-1)} \circledast \cdots \circledast \boldsymbol{U}^{(1)}$
$\boldsymbol{U}^{(n)}$	the $n-th$ factor	$\boldsymbol{U}^{\circledast -n}$	$\boldsymbol{U}^{(N)} \circledast \cdots \circledast \boldsymbol{U}^{(n+1)} \circledast \boldsymbol{U}^{(n-1)} \circledast \cdots \circledast \boldsymbol{U}^{(1)}$
$\boldsymbol{u}_j^{(n)}$	jth column vector of $\boldsymbol{U}^{(n)}$	SIR$(\boldsymbol{a}, \boldsymbol{b})$	$10\log_{10}(\|\boldsymbol{a}\|_2 / \|\boldsymbol{a} - \boldsymbol{b}\|_2)$
$\{\boldsymbol{u}_j\}$	$\{\boldsymbol{u}_j^{(1)}, \boldsymbol{u}_j^{(2)}, \ldots, \boldsymbol{u}_j^{(N)}\}$	PSNR	$20\log_{10}$(Range of Signal/RMSE)
$\underline{\boldsymbol{Y}}$	tensor	Fit$(\underline{\boldsymbol{Y}}, \widehat{\underline{\boldsymbol{Y}}})$	$100(1 - \|\underline{\boldsymbol{Y}} - \widehat{\underline{\boldsymbol{Y}}}\|_F^2 / \|\underline{\boldsymbol{Y}} - \mathrm{E}(\underline{\boldsymbol{Y}})\|_F^2)$

2. Models and Problem Statements

In this paper, we consider at first a simple NMF model described as

Manuscript received July 30, 2008.
Manuscript revised November 11, 2008.
Final manuscript received December 12, 2008.

[†]RIKEN Brain Science Institute, 2-1 Hirosawa, Wako, 351-0198 Saitama, Japan and Warsaw University of Technology and Systems Research Institute, Polish Academy of Science, Poland.
[††]RIKEN Brain Science Institute, 2-1 Hirosawa, Wako, 351-0198 Saitama, Japan.
a) E-mail: cia@brain.riken.jp
b) E-mail: phan@brain.riken.jp

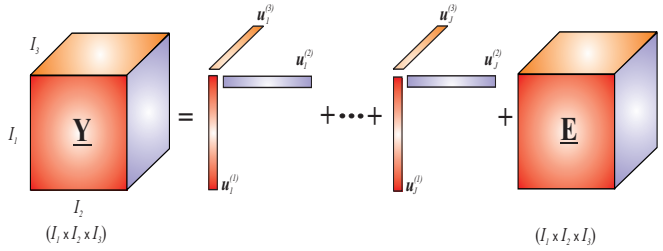


Fig. 1 Illustration of a third-order tensor factorization using standard NTF; Objective is to estimate nonnegative vectors $\boldsymbol{u}_j^{(n)}$ for $j = 1, 2, \ldots, J$ and $n = 1, 2, 3$.

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{E} = \boldsymbol{A}\boldsymbol{B}^T + \boldsymbol{E}, \tag{1}$$

where $\boldsymbol{Y} = [y_{ik}] \in \mathbb{R}^{I \times K}$ is a known input data matrix, $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_J] \in \mathbb{R}_+^{I \times J}$ is an unknown basis (mixing) matrix with nonnegative vectors $\boldsymbol{a}_j \in \mathbb{R}_+^I$, $\boldsymbol{X} = \boldsymbol{B}^T = [\underline{\boldsymbol{x}}_1^T, \underline{\boldsymbol{x}}_2^T, \ldots, \underline{\boldsymbol{x}}_J^T]^T \in \mathbb{R}_+^{J \times K}$ is a matrix representing unknown nonnegative components $\underline{\boldsymbol{x}}_j$ and $\boldsymbol{E} = [e_{ik}] \in \mathbb{R}^{I \times K}$ represents errors or noise. For simplicity, we use also matrix $\boldsymbol{B} = \boldsymbol{X}^T = [\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_J] \in \mathbb{R}_+^{K \times J}$ which allows us to use only column vectors. Our primary objective is to estimate the vectors $\boldsymbol{a}_j$ of the mixing (basis) matrix $\boldsymbol{A}$ and the sources $\underline{\boldsymbol{x}}_j = \boldsymbol{b}_j^T$ (rows of the matrix $\boldsymbol{X}$ or columns of $\boldsymbol{B}$), subject to nonnegativity constraints[†].

The simple NMF model (1) can be naturally extended to the NTF (or nonnegative PARAFAC) as follows: "For a given N-th order tensor $\underline{\boldsymbol{Y}} \in \mathbb{R}^{I_1 \times I_2 \cdots \times I_N}$ perform a nonnegative factorization (decomposition) into a set of N unknown matrices: $\boldsymbol{U}^{(n)} = [\boldsymbol{u}_1^{(n)}, \boldsymbol{u}_2^{(n)}, \ldots, \boldsymbol{u}_J^{(n)}] \in \mathbb{R}_+^{I_n \times J}$, $(n = 1, 2, \ldots, N)$ representing the common (loading) factors", i.e., [11], [16]

$$\underline{\boldsymbol{Y}} = \sum_{j=1}^{J} \boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \cdots \circ \boldsymbol{u}_j^{(N)} + \underline{\boldsymbol{E}} \tag{2}$$

where $\circ$ means outer product of vectors[††] and $\widehat{\underline{\boldsymbol{Y}}} := \sum_{j=1}^{J} \boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \cdots \circ \boldsymbol{u}_j^{(N)}$ is an estimated or approximated (actual) tensor (see Fig. 1). For simplicity, we use the following notations for the parameters of the estimated tensor $\widehat{\underline{\boldsymbol{Y}}} := \sum_{j=1}^{J} [\![\boldsymbol{u}_j^{(1)}, \boldsymbol{u}_j^{(2)}, \ldots, \boldsymbol{u}_j^{(N)}]\!] = [\![\{\boldsymbol{U}\}]\!]$ [16]. A residuum tensor defined as $\underline{\boldsymbol{E}} = \underline{\boldsymbol{Y}} - \widehat{\underline{\boldsymbol{Y}}}$ represents noise or errors depending on applications. This model can be referred to as nonnegative version of CANDECOMP proposed by Carroll and Chang or equivalently nonnegative PARAFAC proposed independently by Harshman and Kruskal. In practice, we usually need to normalize vectors $\boldsymbol{u}_j^{(n)} \in \mathbb{R}^J$ to unit length, i.e., with $\|\boldsymbol{u}_j^{(n)}\|_2 = 1$ for $n = 1, 2, \ldots, N-1$, $\forall j = 1, 2, \ldots, J$, or alternatively apply a Kruskal model:

$$\underline{\boldsymbol{Y}} = \widehat{\underline{\boldsymbol{Y}}} + \underline{\boldsymbol{E}} = \sum_{j=1}^{J} \lambda_j \, (\boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \cdots \circ \boldsymbol{u}_j^{(N)}) \; + \; \underline{\boldsymbol{E}}, \tag{3}$$

where $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \ldots, \lambda_J]^T \in \mathbb{R}_+^J$ are scaling factors and the factors matrices $\boldsymbol{U}^{(n)} = [\boldsymbol{u}_1^{(n)}, \boldsymbol{u}_2^{(n)}, \ldots, \boldsymbol{u}_J^{(n)}]$ have all vectors $\boldsymbol{u}_j^{(n)}$ normalized to unit length columns in the sense $\|\boldsymbol{u}_j^{(n)}\|_2^2 = \boldsymbol{u}_j^{(n)T} \boldsymbol{u}_j^{(n)} = 1, \; \forall j, n$. Generally, the scaling vector $\boldsymbol{\lambda}$ could be derived as $\lambda_j = \|\boldsymbol{u}_j^{(N)}\|_2$. However, we often assume that the weight vector $\boldsymbol{\lambda}$ can be absorbed the (non-normalized) factor matrix $\boldsymbol{U}^{(N)}$, and therefore the model can be expressed in the simplified form (2). The objective is to estimate nonnegative component matrices: $\boldsymbol{U}^{(n)}$ or equivalently the set of vectors $\boldsymbol{u}_j^{(n)}$, $(n = 1, 2, \ldots, N, \quad j = 1, 2, \ldots, J)$, assuming that the number of factors J is known or can be estimated.

It is easy to check that for $N = 2$ and for $\boldsymbol{U}^{(1)} = \boldsymbol{A}$ and $\boldsymbol{U}^{(2)} = \boldsymbol{B} = \boldsymbol{X}^T$ the NTF simplifies to the standard NMF. However, in order to avoid tedious and quite complex notations, we will derive most algorithms first for NMF problem and next attempt to generalize them to the NTF problem, and present basic concepts in clear and easy understandable forms.

Most of known algorithms for the NTF/NMF model are based on alternating least squares (ALS) minimization of the squared Euclidean distance (Frobenius norm) [13], [16], [18]. Especially, for NMF we minimize the following cost function:

$$D_F(\boldsymbol{Y} \,\|\, \widehat{\boldsymbol{Y}}) = \frac{1}{2} \|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{X}\|_F^2, \qquad \widehat{\boldsymbol{Y}} = \boldsymbol{A}\boldsymbol{X}, \tag{4}$$

and for the NTF model (2)

$$D_F(\underline{\boldsymbol{Y}} \,\|\, \widehat{\underline{\boldsymbol{Y}}}) = \frac{1}{2} \left\| \underline{\boldsymbol{Y}} - \sum_{j=1}^{J} (\boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \cdots \circ \boldsymbol{u}_j^{(N)}) \right\|_F^2, \tag{5}$$

subject to nonnegativity constraints and often additional constraints such as sparsity or smoothness [10]. Such formulated problems can be considered as a natural extension of the extensively studied NNLS (Nonnegative Least Squares) formulated as the following optimization problem: "Given a matrix $\boldsymbol{A} \in \mathbb{R}^{I \times J}$ and a set of the observed values given by a vector $\boldsymbol{y} \in \mathbb{R}^I$, find a nonnegative vector $\boldsymbol{x} \in \mathbb{R}^J$ to minimize the cost function $J(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2$, i.e.,

$$\min_{\boldsymbol{x}} J(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\|_2^2, \tag{6}$$

subject to $\boldsymbol{x} \geq \boldsymbol{0}$" [13].

A basic approach to the above formulated optimization problems (4-5) is alternating minimization or alternating projection: The specified cost function is alternately minimized with respect to sets of parameters, each time optimizing one set of arguments while keeping the others fixed. It should be noted that the cost function (4) is convex with respect to entries of $\boldsymbol{A}$ or $\boldsymbol{X}$, but not both. Alternating minimization of the cost function (4) leads to a nonnegative fixed

[†]Usually, a sparsity constraint is naturally and intrinsically provided due to nonlinear projected approach (e.g., half-wave rectifier or adaptive nonnegative shrinkage with gradually decreasing threshold [17]).

[††]For example, the outer product of two vectors $\boldsymbol{a} \in \mathbb{R}^I$, $\boldsymbol{b} \in \mathbb{R}^J$ builds up a rank-one matrix $\boldsymbol{A} = \boldsymbol{a} \circ \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^T \in \mathbb{R}^{I \times J}$ and the outer product of three vectors: $\boldsymbol{a} \in \mathbb{R}^I$, $\boldsymbol{b} \in \mathbb{R}^K$, $\boldsymbol{c} \in \mathbb{R}^Q$ builds up third-order rank-one tensor: $\underline{\boldsymbol{Y}} = \boldsymbol{a} \circ \boldsymbol{b} \circ \boldsymbol{c} \in \mathbb{R}^{I \times K \times Q}$, with entries defined as $y_{ikq} = a_i \, b_k \, c_q$.

point ALS algorithm which can be described briefly as follows:

1. Initialize $\boldsymbol{A}$ randomly or by using the recursive application of Perron-Frobenius theory to SVD [13].
2. Estimate $\boldsymbol{X}$ from the matrix equation $\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{X} = \boldsymbol{A}^T\boldsymbol{Y}$ by solving
$$\min_{\boldsymbol{X}} D_F(\boldsymbol{Y}\|\boldsymbol{A}\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{X}\|_F^2, \text{with fixed } \boldsymbol{A}.$$
3. Set all negative elements of $\boldsymbol{X}$ to zero or a small positive value.
4. Estimate $\boldsymbol{A}$ from the matrix equation $\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{A}^T = \boldsymbol{X}\boldsymbol{Y}^T$ by solving
$$\min_{\boldsymbol{A}} D_F(\boldsymbol{Y}\|\boldsymbol{A}\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{Y}^T - \boldsymbol{X}^T\boldsymbol{A}^T\|_F^2, \text{with fixed } \boldsymbol{X}.$$
5. Set all negative elements of $\boldsymbol{A}$ to zero or a small positive value ε.

The above ALS algorithm can be written in the following explicit form

$$\boldsymbol{X} \leftarrow \max\{\varepsilon, (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{Y}\} := [\boldsymbol{A}^\dagger\boldsymbol{Y}]_+ , \quad (7)$$
$$\boldsymbol{A} \leftarrow \max\{\varepsilon, \boldsymbol{Y}\boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T)^{-1}\} := [\boldsymbol{Y}\boldsymbol{X}^\dagger]_+ , \quad (8)$$

where $\boldsymbol{A}^\dagger$ is the Moore-Penrose inverse of $\boldsymbol{A}$, ε is a small constant (typically, 10^{-16}) to enforce positive entries. Note that `max` operator is performed component-wise for entries of matrices. Various additional constraints on $\boldsymbol{A}$ and $\boldsymbol{X}$ can be imposed [19].

For large scale NMF problem for $J << I$ and $J << K$ the data matrix $\boldsymbol{Y}$ is usually low rank and in such cases we do not need to process all elements of the matrix in order to estimate factor matrices $\boldsymbol{A}$ and $\boldsymbol{X}$ (see Fig. 2). In fact, instead of performing large scale factorization of (1), we can consider alternating factorization of much smaller dimension problems:

$$\boldsymbol{Y}_r = \boldsymbol{A}_r\boldsymbol{X} + \boldsymbol{E}_r, \quad \text{for fixed (known)} \quad \boldsymbol{A}_r, \quad (9)$$
$$\boldsymbol{Y}_c = \boldsymbol{A}\boldsymbol{X}_c + \boldsymbol{E}_c, \quad \text{for fixed (known)} \quad \boldsymbol{X}_c, \quad (10)$$

where $\boldsymbol{Y}_r \in \mathbb{R}_+^{R\times K}$ and $\boldsymbol{Y}_c \in \mathbb{R}_+^{I\times C}$ are matrices constructed form preselected rows and columns of the data matrix $\boldsymbol{Y}$, respectively. Analogously, we construct reduced dimensions matrices: $\boldsymbol{A}_r \in \mathbb{R}^{R\times J}$ and $\boldsymbol{X}_c \in \mathbb{R}^{J\times C}$ by using the same indexes for columns and rows which were used for construction of the matrices $\boldsymbol{Y}_c$ and $\boldsymbol{Y}_r$, respectively. There are several strategies to chose columns and rows of the input matrix data. The simplest scenario is to chose the first R rows and the first C columns of data matrix $\boldsymbol{Y}$. Alternatively, we can select randomly, e.g., uniformly distributed, i.e. every N row and column. Another option is to chose such rows and columns that provide the largest ℓ_p-norm values. For noisy data with uncorrelated noise, we can construct new columns and rows as local average (mean values) of some specific numbers of columns and rows of the raw data. For example, the first selected column is created as average of the first M columns, the second columns is average of the next

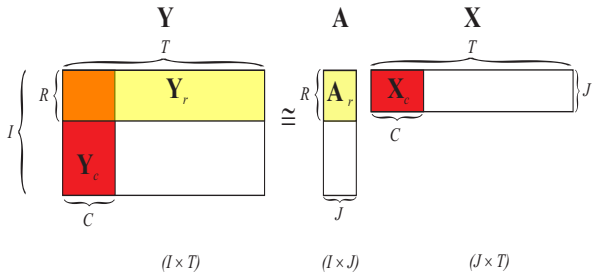


Fig. 2 Conceptual illustration of processing of data for a large scale NMF. Instead of processing the whole matrix $\boldsymbol{Y} \in \mathbb{R}^{I\times K}$, we process much smaller dimensional block matrices $\boldsymbol{Y}_c \in \mathbb{R}^{I\times C}$ and $\boldsymbol{Y}_r \in \mathbb{R}^{R\times K}$ and corresponding factor matrices $\boldsymbol{X}_c \in \mathbb{R}^{J\times C}$ and $\boldsymbol{A}_r \in \mathbb{R}^{R\times J}$ with $C << K$ and $R << I$. For simplicity, we have assumed that the first R rows and the first C columns of the matrices $\boldsymbol{Y}$, $\boldsymbol{A}$, $\boldsymbol{X}$ are chosen, respectively.

M columns, and so on. The same procedure is applied for rows. Another approach is to cluster all columns and rows in C and R cluster and select one column and one row form each cluster, respectively. In practice, it is sufficient to chose $J < R \le 4J$ and $J < C \le 4J$. In the special case, for squared Euclidean distance (Frobenius norm) instead of alternating minimizing the cost function:

$$D_F(\boldsymbol{Y} \,\|\, \boldsymbol{A}\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{X}\|_F^2 ,$$

we can minimize sequentially two cost functions:

$$D_F(\boldsymbol{Y}_r \,\|\, \boldsymbol{A}_r\boldsymbol{X}) = \frac{1}{2}\|\boldsymbol{Y}_r - \boldsymbol{A}_r\boldsymbol{X}\|_F^2 , \quad \text{for fixed} \quad \boldsymbol{A}_r,$$
$$D_F(\boldsymbol{Y}_c \,\|\, \boldsymbol{A}\boldsymbol{X}_c) = \frac{1}{2}\|\boldsymbol{Y}_c - \boldsymbol{A}\boldsymbol{X}_c\|_F^2 , \quad \text{for fixed} \quad \boldsymbol{X}_c.$$

Minimization of these cost functions with respect to $\boldsymbol{X}$ and $\boldsymbol{A}$, subject to nonnegativity constraints leads to simple ALS update formulas for the large scale NMF:

$$\boldsymbol{A} \leftarrow \left[\boldsymbol{Y}_c\boldsymbol{X}_c^\dagger\right]_+ = \left[\boldsymbol{Y}_c\boldsymbol{X}_c^T(\boldsymbol{X}_c\boldsymbol{X}_c^T)^{-1}\right]_+ , \quad (11)$$
$$\boldsymbol{X} \leftarrow \left[\boldsymbol{A}_r^\dagger\boldsymbol{Y}_r\right]_+ = \left[(\boldsymbol{A}_r^T\boldsymbol{A}_r)^{-1}\boldsymbol{A}_r^T\boldsymbol{Y}_r\right]_+ . \quad (12)$$

The nonnegative ALS algorithm can be generalized for the NTF problem (2) [16]:

$$\boldsymbol{U}^{(n)} \leftarrow \left[\boldsymbol{Y}_{(n)}\,\boldsymbol{U}^{\odot_{-n}}\left(\boldsymbol{U}^{\odot_{-n}T}\boldsymbol{U}^{\odot_{-n}}\right)^{-1}\right]_+ ,$$
$$= \left[\boldsymbol{Y}_{(n)}\,\boldsymbol{U}^{\odot_{-n}}\left(\left\{\boldsymbol{U}^T\boldsymbol{U}\right\}^{\circledast_{-n}}\right)^{-1}\right]_+ , \; n = 1, \ldots, N. \quad (13)$$

where $\boldsymbol{Y}_{(n)} \in \mathbb{R}_+^{I_n\times I_1\cdots I_{n-1}I_{n+1}\cdots I_N}$ is n-mode unfolded matrix of the tensor $\underline{\boldsymbol{Y}} \in \mathbb{R}_+^{I_1\times I_2\times\cdots\times I_N}$ and $\left\{\boldsymbol{U}^T\boldsymbol{U}\right\}^{\circledast_{-n}} = (\boldsymbol{U}^{(N)T}\boldsymbol{U}^{(N)}) \circledast \cdots \circledast (\boldsymbol{U}^{(n+1)T}\boldsymbol{U}^{(n+1)}) \circledast (\boldsymbol{U}^{(n-1)T}\boldsymbol{U}^{(n-1)}) \circledast \cdots \circledast (\boldsymbol{U}^{(1)T}\boldsymbol{U}^{(1)})$.

At present, ALS algorithms for NMF and NTF are considered as "workhorse" approaches, however they may take many iterations to converge. Moreover, they are also not guaranteed to converge to a global minimum or even a stationary point, but only to a solution where the cost functions cease to decrease [13], [16]. However, the ALS method can be considerably improved and the computational complexity reduced as will be shown in this paper.

In fact, in this paper, we use a different and more sophisticated approach. Instead of minimizing one or two cost functions, we minimize a set of local cost functions with the same global minima (e.g., squared Euclidean distances and Alpha or Beta divergences with a single parameter alpha or beta). The majority of known algorithms for NMF work only if the following assumption $K >> I \geq J$ is satisfied, where J is the number of the nonnegative components. The NMF algorithms developed in this paper are suitable also for the under-determined case, i.e., for $K > J > I$, if sources are sparse enough. Moreover, the proposed algorithms are robust with respect to noise and suitable for large scale problems. Furthermore, in this paper we consider the extension of our approach to NMF/NTF models with optional sparsity and smoothness constraints.

3. Derivation of Fast HALS NMF Algorithms

Denoting the columns by $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_J]$ and $\boldsymbol{B} = [\boldsymbol{b}_1, \boldsymbol{b}_2, \ldots, \boldsymbol{b}_J]$, we can express the squared Euclidean cost function as

$$J(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_J, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_J) = \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{B}^T\|_F^2 = \frac{1}{2}\|\boldsymbol{Y} - \sum_{j=1}^{J} \boldsymbol{a}_j \boldsymbol{b}_j^T\|_F^2. \quad (14)$$

The basic idea is to define residues:

$$\boldsymbol{Y}^{(j)} = \boldsymbol{Y} - \sum_{p \neq j} \boldsymbol{a}_p \boldsymbol{b}_p^T = \boldsymbol{Y} - \boldsymbol{A}\boldsymbol{B}^T + \boldsymbol{a}_j \boldsymbol{b}_j^T, = \boldsymbol{Y} - \boldsymbol{A}\boldsymbol{B}^T + \boldsymbol{a}_{j-1}\boldsymbol{b}_{j-1}^T - \boldsymbol{a}_{j-1}\boldsymbol{b}_{j-1}^T + \boldsymbol{a}_j \boldsymbol{b}_j^T \quad (15)$$

for $j = 1, 2, \ldots, J$ and minimize alternatively the set of cost functions (with respect to set of parameters $\{\boldsymbol{a}_j\}$ and $\{\boldsymbol{b}_j\}$):

$$D_A^{(j)}(\boldsymbol{a}_j) = \frac{1}{2}\|\boldsymbol{Y}^{(j)} - \boldsymbol{a}_j \boldsymbol{b}_j^T\|_F^2, \text{ for a fixed } \boldsymbol{b}_j, \quad (16)$$

$$D_B^{(j)}(\boldsymbol{b}_j) = \frac{1}{2}\|\boldsymbol{Y}^{(j)} - \boldsymbol{a}_j \boldsymbol{b}_j^T\|_F^2, \text{ for a fixed } \boldsymbol{a}_j, \quad (17)$$

for $j = 1, 2, \ldots, J$ subject to $\boldsymbol{a}_j \geq \boldsymbol{0}$ and $\boldsymbol{b}_j \geq \boldsymbol{0}$, respectively.

In other words, we minimize alternatively the set of cost functions

$$D_F^{(j)}(\boldsymbol{Y}^{(j)} \| \boldsymbol{a}_j \boldsymbol{b}_j^T) = \frac{1}{2}\|\boldsymbol{Y}^{(j)} - \boldsymbol{a}_j \boldsymbol{b}_j^T\|_F^2, \quad (18)$$

for $j = 1, 2, \ldots, J$ subject to $\boldsymbol{a}_j \geq \boldsymbol{0}$ and $\boldsymbol{b}_j \geq \boldsymbol{0}$, respectively.

The gradients of the local cost functions (18) with respect to the unknown vectors $\boldsymbol{a}_j$ and $\boldsymbol{b}_j$ (assuming that other vectors are fixed) are expressed by

$$\frac{\partial D_F^{(j)}(\boldsymbol{Y}^{(j)} \| \boldsymbol{a}_j \boldsymbol{b}_j^T)}{\partial \boldsymbol{a}_j} = \boldsymbol{a}_j \boldsymbol{b}_j^T \boldsymbol{b}_j - \boldsymbol{Y}^{(j)} \boldsymbol{b}_j, \quad (19)$$

$$\frac{\partial D_F^{(j)}(\boldsymbol{Y}^{(j)} \| \boldsymbol{a}_j \boldsymbol{b}_j^T)}{\partial \boldsymbol{b}_j} = \boldsymbol{b}_j \boldsymbol{a}_j^T \boldsymbol{a}_j - \boldsymbol{Y}^{(j)T} \boldsymbol{a}_j. \quad (20)$$

By equating the gradient components to zero and assuming that we enforce the nonnegativity constraints with a simple "half-wave rectifying" nonlinear projection, we obtain a simple set of sequential learning rules:

$$\boldsymbol{b}_j \leftarrow \frac{1}{\boldsymbol{a}_j^T \boldsymbol{a}_j}\left[\boldsymbol{Y}^{(j)T} \boldsymbol{a}_j\right]_+, \quad \boldsymbol{a}_j \leftarrow \frac{1}{\boldsymbol{b}_j^T \boldsymbol{b}_j}\left[\boldsymbol{Y}^{(j)} \boldsymbol{b}_j\right]_+, \quad (21)$$

for $j = 1, 2, \ldots, J$. We refer to these update rules as the HALS algorithm which we first introduced in [3]. The same or similar update rules for the NMF have been proposed or rediscovered independently in [20]–[23]. However, our practical implementations of the HALS algorithm are quite different and allow various extensions to sparse and smooth NMF, and also for the N-order NTF.

First of all, from the formula (15) it follows that we do not need to compute explicitly the residue matrix $\boldsymbol{Y}^{(j)}$ in each iteration step but just smartly update it [24].

It is interesting to note that such nonlinear projections can be imposed individually for each source $\underline{\boldsymbol{x}}_j$ and/or vector $\boldsymbol{a}_j$, so the algorithm can be directly extended to a semi-NMF or a semi-NTF model in which some parameters are relaxed to be bipolar (by removing the half-wave rectifying operator $[\cdot]_+$, if necessary). Furthermore, in practice, it is necessary to normalize in each iteration step the column vectors $\boldsymbol{a}_j$ and/or $\boldsymbol{b}_j$ to unit length vectors (in the sense of ℓ_p-norm ($p = 1, 2, ..., \infty$)). In the special case of ℓ_2-norm, the above algorithm can be further simplified by ignoring denominators in (21) and imposing normalization of vectors after each iteration steps. The standard HALS local updating rules can be written in a simplified scalar form:

$$b_{kj} \leftarrow \left[\sum_{i=1}^{I} a_{ij}\, y_{ik}^{(j)}\right]_+, \quad a_{ij} \leftarrow \left[\sum_{k=1}^{K} b_{kj}\, y_{ik}^{(j)}\right]_+, \quad (22)$$

with $a_{ij} \leftarrow a_{ij}/\|\boldsymbol{a}_j\|_2$, where $y_{ik}^{(j)} = [\boldsymbol{Y}^{(j)}]_{ik} = y_{ik} - \sum_{p \neq j} a_{ip} b_{kp}$. Efficient implementation of the HALS algorithm (22) is illustrated by detailed pseudo-code given in Algorithm 1.

Algorithm 1 HALS for NMF: Given $\boldsymbol{Y} \in \mathbb{R}_+^{I \times K}$ estimate $\boldsymbol{A} \in \mathbb{R}_+^{I \times J}$ and $\boldsymbol{X} = \boldsymbol{B}^T \in \mathbb{R}_+^{J \times K}$

```
Initialize nonnegative matrix A and/or X = B^T using ALS
Normalize the vectors a_j (or b_j) to unit ℓ2-norm length,
E = Y − AB^T;
repeat
    for j = 1 to J do
        Y^(j) ⇐ E + a_j b_j^T;
        b_j ⇐ [Y^(j)T a_j]_+
        a_j ⇐ [Y^(j) b_j]_+
        a_j ⇐ a_j/‖a_j‖_2;
        E ⇐ Y^(j) − a_j b_j^T;
    end for
until convergence criterion is reached
```

3.1 Extensions and Practical Implementations of Fast HALS

The above simple algorithm can be further extended or improved (in respect of convergence rate and performance

and by imposing additional constraints such as sparsity and smoothness). First of all, different cost functions can be used for estimation of the rows of the matrix $\boldsymbol{X} = \boldsymbol{B}^T$ and the columns of the matrix $\boldsymbol{A}$ (possibly with various additional regularization terms [19], [25]). Furthermore, the columns of $\boldsymbol{A}$ can be estimated simultaneously, instead of one by one. For example, by minimizing the set of cost functions in (4) with respect to $\boldsymbol{b}_j$, and simultaneously the cost function (18) with normalization of the columns $\boldsymbol{a}_j$ to unit ℓ_2-norm, we obtain a very efficient NMF learning algorithm in which the individual vectors of $\boldsymbol{B}$ are updated locally (column by column) and the matrix $\boldsymbol{A}$ is updated globally using nonnegative ALS (all columns $\boldsymbol{a}_j$ simultaneously) (see also [19]):

$$\boldsymbol{b}_j \leftarrow \left[\boldsymbol{Y}_r^{(j)\,T}\,\tilde{\boldsymbol{a}}_j\right]_+ /(\tilde{\boldsymbol{a}}_j^T\,\tilde{\boldsymbol{a}}_j), \quad \boldsymbol{A} \leftarrow \left[\boldsymbol{Y}_c\boldsymbol{X}_c^T(\boldsymbol{X}_c\boldsymbol{X}_c^T)^{-1}\right]_+, \tag{23}$$

where $\tilde{\boldsymbol{a}}_j$ is an j-th vector of a reduced matrix $\boldsymbol{A}_r \in \mathbb{R}_+^{R\times J}$. Matrix $\boldsymbol{A}$ needs to be normalized to the unit length column vectors in the ℓ_2-norm sense after each iteration.

Alternatively, even more efficient approach is to perform factor by factor procedure, instead of updating column-by column vectors [24]. From (21), we obtain the following update rule for $\boldsymbol{b}_j = \underline{\boldsymbol{x}}_j^T$

$$\begin{aligned}\boldsymbol{b}_j &\leftarrow \boldsymbol{Y}^{(j)T}\boldsymbol{a}_j/(\boldsymbol{a}_j^T\boldsymbol{a}_j) = \left(\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{B}^T + \boldsymbol{a}_j\boldsymbol{b}_j^T\right)^T \boldsymbol{a}_j/(\boldsymbol{a}_j^T\boldsymbol{a}_j)\\ &= (\boldsymbol{Y}^T\boldsymbol{a}_j - \boldsymbol{B}\boldsymbol{A}^T\boldsymbol{a}_j + \boldsymbol{b}_j\boldsymbol{a}_j^T\boldsymbol{a}_j)/(\boldsymbol{a}_j^T\boldsymbol{a}_j),\\ &= \left(\left[\boldsymbol{Y}^T\boldsymbol{A}\right]_j - \boldsymbol{B}\left[\boldsymbol{A}^T\boldsymbol{A}\right]_j + \boldsymbol{b}_j\boldsymbol{a}_j^T\boldsymbol{a}_j\right)/(\boldsymbol{a}_j^T\boldsymbol{a}_j),\end{aligned} \tag{24}$$

with $\boldsymbol{b}_j \leftarrow \left[\boldsymbol{b}_j\right]_+$. Due to $\|\boldsymbol{a}_j\|_2^2 = 1$, the learning rule for $\boldsymbol{b}_j$ has a simplified form

$$\boldsymbol{b}_j \leftarrow \left[\boldsymbol{b}_j + \left[\boldsymbol{Y}^T\boldsymbol{A}\right]_j - \boldsymbol{B}\left[\boldsymbol{A}^T\boldsymbol{A}\right]_j\right]_+. \tag{25}$$

Analogously to equation (24), the learning rule for $\boldsymbol{a}_j$ is given by

$$\boldsymbol{a}_j \leftarrow \left[\boldsymbol{a}_j\boldsymbol{b}_j^T\boldsymbol{b}_j + [\boldsymbol{Y}\boldsymbol{B}]_j - \boldsymbol{A}\left[\boldsymbol{B}^T\boldsymbol{B}\right]_j\right]_+, \tag{26}$$

$$\boldsymbol{a}_j \leftarrow \boldsymbol{a}_j/\|\boldsymbol{a}_j\|_2. \tag{27}$$

Based on these expressions, we have designed and implemented the improved and modified HALS algorithm given below in the pseudo-code as Algorithm 2. For large scale data and block-wise strategy, the fast HALS learning rule for $\boldsymbol{b}_j$ is rewritten from (24) as follows

$$\begin{aligned}\boldsymbol{b}_j &\leftarrow \left[\boldsymbol{b}_j + \left[\boldsymbol{Y}_r^T\boldsymbol{A}_r\right]_j/\|\tilde{\boldsymbol{a}}_j\|_2^2 - \boldsymbol{B}\left[\boldsymbol{A}_r^T\boldsymbol{A}_r\right]_j/\|\tilde{\boldsymbol{a}}_j\|_2^2\right]_+\\ &= \left[\boldsymbol{b}_j + \left[\boldsymbol{Y}_r^T\boldsymbol{A}_r\boldsymbol{D}_{A_r}\right]_j - \boldsymbol{B}\left[\boldsymbol{A}_r^T\boldsymbol{A}_r\boldsymbol{D}_{A_r}\right]_j\right]_+\end{aligned} \tag{28}$$

where $\boldsymbol{D}_{A_r} = \mathrm{diag}(\|\tilde{\boldsymbol{a}}_1\|_2^{-2}, \|\tilde{\boldsymbol{a}}_2\|_2^{-2}, \ldots, \|\tilde{\boldsymbol{a}}_J\|_2^{-2})$ is a diagonal matrix. The learning rule for $\boldsymbol{a}_j$ has a similar form

$$\boldsymbol{a}_j \leftarrow \left[\boldsymbol{a}_j + [\boldsymbol{Y}_c\boldsymbol{B}_c\boldsymbol{D}_{B_c}]_j - \boldsymbol{A}\left[\boldsymbol{B}_c^T\boldsymbol{B}_c\boldsymbol{D}_{B_c}\right]_j\right]_+ \tag{29}$$

where $\boldsymbol{D}_{B_c} = \mathrm{diag}(\|\tilde{\boldsymbol{b}}_1\|_2^{-2}, \|\tilde{\boldsymbol{b}}_2\|_2^{-2}, \ldots, \|\tilde{\boldsymbol{b}}_J\|_2^{-2})$ and $\tilde{\boldsymbol{b}}_j$ is the j-th vector of the reduced matrix $\boldsymbol{B}_c = \boldsymbol{X}_c^T \in \mathbb{R}_+^{C\times J}$.

Algorithm 2 FAST HALS for NMF: $\boldsymbol{Y} \approx \boldsymbol{A}\boldsymbol{B}^T$

1: Initialize nonnegative matrix $\boldsymbol{A}$ and/or $\boldsymbol{B}$ using ALS
2: Normalize the vectors $\boldsymbol{a}_j$ (or $\boldsymbol{b}_j$) to unit ℓ_2-norm length
3: **repeat**
4: % Update $\boldsymbol{B}$;
5: $\boldsymbol{W} = \boldsymbol{Y}^T\boldsymbol{A}$;
6: $\boldsymbol{V} = \boldsymbol{A}^T\boldsymbol{A}$;
7: **for** $j = 1$ to J **do**
8: $\boldsymbol{b}_j \Leftarrow \left[\boldsymbol{b}_j + \boldsymbol{w}_j - \boldsymbol{B}\,\boldsymbol{v}_j\right]_+$
9: **end for**
10: % Update $\boldsymbol{A}$;
11: $\boldsymbol{P} = \boldsymbol{Y}\boldsymbol{B}$;
12: $\boldsymbol{Q} = \boldsymbol{B}^T\boldsymbol{B}$;
13: **for** $j = 1$ to J **do**
14: $\boldsymbol{a}_j \Leftarrow \left[\boldsymbol{a}_j\,q_{jj} + \boldsymbol{p}_j - \boldsymbol{A}\,\boldsymbol{q}_j\right]_+$
15: $\boldsymbol{a}_j \Leftarrow \boldsymbol{a}_j/\|\boldsymbol{a}_j\|_2$;
16: **end for**
17: **until** convergence criterion is reached

3.2 HALS NMF Algorithm with Sparsity and Smoothness Constraints

In order to impose sparseness and smoothness constraints for vectors $\boldsymbol{b}_j$ (source signals), we can minimize the following set of cost functions:

$$\begin{aligned}D_F^{(j)}(\boldsymbol{Y}^{(j)}\|\boldsymbol{a}_j\,\boldsymbol{b}_j^T) = \frac{1}{2}\|\boldsymbol{Y}^{(j)} - \boldsymbol{a}_j\boldsymbol{b}_j^T\|_F^2 +\\ +\alpha_{sp}\,\|\boldsymbol{b}_j\|_1 + \alpha_{sm}\,\|\varphi(\boldsymbol{L}\,\boldsymbol{b}_j)\|_1,\end{aligned} \tag{30}$$

for $j = 1, 2, \ldots, J$ subject to $\boldsymbol{a}_j \geq \boldsymbol{0}$ and $\boldsymbol{b}_j \geq \boldsymbol{0}$, where $\alpha_{sp} > 0$, $\alpha_{sm} > 0$ are regularization parameters controlling level of sparsity and smoothness, respectively, $\boldsymbol{L}$ is a suitably designed matrix (the Laplace operator) which measures the smoothness (by estimating the differences between neighboring samples of $\boldsymbol{b}_j$)[†] and $\varphi : \mathbb{R} \to \mathbb{R}$ is an edge-preserving function applied componentwise. Although this edge-preserving nonlinear function may take various forms [26]:

$$\varphi(t) = |t|^\alpha/\alpha, \quad 1 \leq \alpha \leq 2, \tag{31}$$

$$\varphi(t) = \sqrt{\alpha + t^2}, \tag{32}$$

$$\varphi(t) = 1 + |t|/\alpha - \log(1 + |t|/\alpha), \quad \alpha > 0, \tag{33}$$

we restrict ourself to simple cases, where $\varphi(t) = |t|^\alpha/\alpha$ for $\alpha = 1$ or 2, and $\boldsymbol{L}$ is the derivative operator of the first or second order. For example, the first order derivative operator $\boldsymbol{L}$ with K points can take the form:

$$\boldsymbol{L} = \begin{pmatrix} 1 & -1 & & & \\ & 1 & -1 & & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \end{pmatrix} \tag{34}$$

and the cost function (30) becomes similar to the total-variation (TV) regularization (which is often used in signal and image recovery) but with additional sparsity constraints:

[†]In the special case for $\boldsymbol{L} = \boldsymbol{I}_K$ and $\varphi(t) = |t|$, the smoothness regularization term becomes sparsity term.

$$D_F^{(j)}(\boldsymbol{Y}^{(j)}\|\boldsymbol{a}_j\,\boldsymbol{b}_j^T) = \frac{1}{2}\left\|\boldsymbol{Y}^{(j)} - \boldsymbol{a}_j\boldsymbol{b}_j^T\right\|_F^2 + \alpha_{sp}\,\|\boldsymbol{b}_j\|_1 + + \alpha_{sm}\sum_{k=1}^{K-1}|b_{k\,j} - b_{(k+1)\,j}|. \quad (35)$$

Another important case assumes that $\varphi(t) = \frac{1}{2}|t|^2$ and $\boldsymbol{L}$ is the second order derivative operator with K points. In such a case, we obtain the Tikhonov-like regularization:

$$D_F^{(j)}(\boldsymbol{Y}^{(j)}\|\boldsymbol{a}_j\,\boldsymbol{b}_j^T) = \frac{1}{2}\|\boldsymbol{Y}^{(j)} - \boldsymbol{a}_j\boldsymbol{b}_j^T\|_F^2 + \alpha_{sp}\,\|\boldsymbol{b}_j\|_1 + + \frac{1}{2}\,\alpha_{sm}\,\|\boldsymbol{L}\boldsymbol{b}_j\|_2^2. \quad (36)$$

In the such case the update rule for $\boldsymbol{a}_j$ is the same as in (21), whereas the update rule for $\boldsymbol{b}_j$ is given by:

$$\boldsymbol{b}_j \leftarrow (\boldsymbol{I} + \alpha_{sm}\,\boldsymbol{L}^T\boldsymbol{L})^{-1}(\boldsymbol{Y}^{(j)\,T}\boldsymbol{a}_j - \alpha_{sp}\,\mathbf{1}_K). \quad (37)$$

where $\mathbf{1}_K \in \mathbb{R}^K$ is a vector with all one. This learning rule is robust to noise, however, it involves a rather high computational cost due to the calculation of an inverse of a large matrix in each iteration. To circumvent this problem and to considerably reduce the complexity of the algorithm we present a second-order smoothing operator $\boldsymbol{L}$ in the following form:

$$\boldsymbol{L} = \begin{pmatrix} -2 & 2 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & \ddots & \ddots & & & \\ & & & 1 & -2 & 1 & \\ & & & & 2 & -2 & \end{pmatrix}$$

$$= \begin{pmatrix} -2 & & & & & & \\ & -2 & & & & & \\ & & -2 & & & & \\ & & & \ddots & & & \\ & & & & -2 & & \\ & & & & & -2 & \end{pmatrix} + \begin{pmatrix} 0 & 2 & & & & & \\ 1 & 0 & 1 & & & & \\ & 1 & 0 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 0 & 1 & \\ & & & & 2 & 0 & \end{pmatrix}$$

$$= -2\boldsymbol{I} + 2\boldsymbol{S}. \quad (38)$$

However, instead of computing directly $\boldsymbol{L}\boldsymbol{b}_j = -2\boldsymbol{I}\boldsymbol{b}_j + 2\boldsymbol{S}\boldsymbol{b}_j$, in the second term we replace $\boldsymbol{b}_j$ by its estimation $\hat{\boldsymbol{b}}_j$ obtained from the previous update. Hence, a new smoothing regularization term with $\varphi(t) = t^2/8$ takes a simplified and computationally more efficient form

$$J_{sm} = \|\varphi(\,-2\boldsymbol{b}_j + 2\boldsymbol{S}\hat{\boldsymbol{b}}_j)\|_1 = \frac{1}{2}\|\boldsymbol{b}_j - \boldsymbol{S}\hat{\boldsymbol{b}}_j\|_2^2\,. \quad (39)$$

Finally, the learning rule of the regularized HALS algorithm takes the following form:

$$\boldsymbol{b}_j \leftarrow \left[\boldsymbol{Y}^{(j)T}\,\boldsymbol{a}_j - \alpha_{sp}\,\mathbf{1}_K + \alpha_{sm}\,\boldsymbol{S}\,\hat{\boldsymbol{b}}_j\right]_+ /(\boldsymbol{a}_j^T\,\boldsymbol{a}_j + \alpha_{sm}) = \left[\boldsymbol{Y}^{(j)T}\,\boldsymbol{a}_j - \alpha_{sp}\,\mathbf{1}_K + \alpha_{sm}\,\boldsymbol{S}\,\hat{\boldsymbol{b}}_j\right]_+ /(1 + \alpha_{sm})\,. \quad (40)$$

Alternatively, for a relatively small dimension of matrix $\boldsymbol{A}$, an efficient solution is based on a combination of a local learning rule for the vectors of $\boldsymbol{B}$ and a global one for $\boldsymbol{A}$, based on the nonnegative ALS algorithm:

$$\boldsymbol{b}_j \leftarrow \left[\boldsymbol{Y}^{(j)T}\boldsymbol{a}_j - \alpha_{sp}\,\mathbf{1}_K + \alpha_{sm}\,\boldsymbol{S}\hat{\boldsymbol{b}}_j\right]_+ /(1 + \alpha_{sm}),$$
$$\boldsymbol{A} \leftarrow \left[\boldsymbol{Y}_c\boldsymbol{X}_c^T(\boldsymbol{X}_c\boldsymbol{X}_c^T)^{-1}\right]_+, \quad (41)$$

with the normalization (scaling) of the columns of $\boldsymbol{A}$ to the unit length ℓ_2-norm.

An important open problem is an optimal choice of regularization parameters α_{sm}. Selection of appropriate regularization parameters plays a key role. Similar to the Tikhonov-like regularization approach we selected an optimal α_{sm} by applying the L-curve technique [27] to estimate a corner of the L-curve. However, in the NMF, since both matrices $\boldsymbol{A}$ and $\boldsymbol{X}$ are unknown, the procedure is slightly different: first, we initiate $\alpha_{sm} = 0$ and perform a preliminary update to obtain $\boldsymbol{A}$ and $\boldsymbol{X}$; next we set α_{sm} by the L-curve corner based on the preliminary estimated matrix $\boldsymbol{A}$; then, we continue updating until convergence is achieved.

4. Fast HALS NTF Algorithm Using Squared Euclidean Distances

The above approaches can be relatively easily extended to the NTF problem. Let us consider sequential minimization of a set of local cost functions:

$$D_F^{(j)}(\underline{\boldsymbol{Y}}^{(j)}\|\widehat{\underline{\boldsymbol{Y}}}^{(j)}) = \frac{1}{2}\left\|\underline{\boldsymbol{Y}}^{(j)} - \boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \cdots \circ \boldsymbol{u}_j^{(N)}\right\|_F^2 \quad (42)$$

$$= \frac{1}{2}\left\|\boldsymbol{Y}_{(n)}^{(j)} - \boldsymbol{u}_j^{(n)}\,\left\{\boldsymbol{u}_j\right\}^{\odot_{-n}T}\right\|_F^2, \quad (43)$$

for $j = 1, 2, \ldots, J$, subject to the nonnegativity constraints, where $\widehat{\underline{\boldsymbol{Y}}}^{(j)} = \boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \cdots \circ \boldsymbol{u}_j^{(N)}$, $\left\{\boldsymbol{u}_j\right\}^{\odot_{-n}T} = [\boldsymbol{u}_j^{(N)}]^T \odot \cdots \odot [\boldsymbol{u}_j^{(n+1)}]^T \odot [\boldsymbol{u}_j^{(n-1)}]^T \odot \cdots \odot [\boldsymbol{u}_j^{(1)}]^T$ and

$$\underline{\boldsymbol{Y}}^{(j)} = \underline{\boldsymbol{Y}} - \sum_{p\neq j}\boldsymbol{u}_p^{(1)} \circ \boldsymbol{u}_p^{(2)} \circ \cdots \circ \boldsymbol{u}_p^{(N)} \quad (44)$$

$$= \underline{\boldsymbol{Y}} - \sum_{p=1}^{J}(\boldsymbol{u}_p^{(1)} \circ \cdots \circ \boldsymbol{u}_p^{(N)}) + (\boldsymbol{u}_j^{(1)} \circ \cdots \circ \boldsymbol{u}_j^{(N)})$$

$$= \underline{\boldsymbol{Y}} - \widehat{\underline{\boldsymbol{Y}}} + [\![\{\boldsymbol{u}_j\}]\!]. \quad (45)$$

where $[\![\{\boldsymbol{u}_j\}]\!] = \boldsymbol{u}_j^{(1)} \circ \cdots \circ \boldsymbol{u}_j^{(N)}$ is a rank-one tensor. Note that (43) is the $n-$mode matricized (unfolded) version of (42). The gradients of (43) with respect to elements $\boldsymbol{u}_j^{(n)}$ are given by

$$\frac{\partial D_F^{(j)}}{\partial \boldsymbol{u}_j^{(n)}} = -\boldsymbol{Y}_{(n)}^{(j)}\left\{\boldsymbol{u}_j\right\}^{\odot_{-n}} + \boldsymbol{u}_j^{(n)}\left\{\boldsymbol{u}_j\right\}^{\odot_{-n}T}\left\{\boldsymbol{u}_j\right\}^{\odot_{-n}} \quad (46)$$

$$= -\boldsymbol{Y}_{(n)}^{(j)}\left\{\boldsymbol{u}_j\right\}^{\odot_{-n}} + \boldsymbol{u}_j^{(n)}\gamma_j^{(n)}, \quad (47)$$

where scaling coefficients $\gamma_j^{(n)}$ can be computed as follows:

$$\gamma_j^{(n)} = \left\{\boldsymbol{u}_j\right\}^{\odot_{-n}T}\left\{\boldsymbol{u}_j\right\}^{\odot_{-n}} = \left\{\boldsymbol{u}_j^T\boldsymbol{u}_j\right\}^{\circledast_{-n}} = \left\{\boldsymbol{u}_j^T\boldsymbol{u}_j\right\}^{\circledast} / \left(\boldsymbol{u}_j^{(n)T}\boldsymbol{u}_j^{(n)}\right) = \left(\boldsymbol{u}_j^{(N)T}\boldsymbol{u}_j^{(N)}\right) / \left(\boldsymbol{u}_j^{(n)T}\boldsymbol{u}_j^{(n)}\right) = \begin{cases}\boldsymbol{u}_j^{(N)T}\boldsymbol{u}_j^{(N)}, & n \neq N \\ 1, & n = N.\end{cases} \quad (48)$$

Hence, a new HALS NTF learning rule for $\boldsymbol{u}_j^{(n)}$, ($j =$

$1, 2, \ldots, N$; $n = 1, 2, \ldots, N$) is obtained by equating the gradient (47) to zero:

$$\boldsymbol{u}_j^{(n)} \leftarrow \boldsymbol{Y}_{(n)}^{(j)} \left\{\boldsymbol{u}_j\right\}^{\odot_{-n}}. \tag{49}$$

Note that the scaling factors $\gamma_j^{(n)}$ have been ignored due to normalization after each iteration step $\boldsymbol{u}_j^{(n)} = \boldsymbol{u}_j^{(n)}/\|\boldsymbol{u}_j^{(n)}\|_2$ for $n = 1, 2, \ldots N-1$. The learning rule (49) can be written in an equivalent form expressed by n mode multiplication of tensor by vectors:

$$\begin{aligned}\boldsymbol{u}_j^{(n)} &\leftarrow \underline{\boldsymbol{Y}}^{(j)} \,\overline{\times}_1 \boldsymbol{u}_j^{(1)} \cdots \overline{\times}_{n-1} \boldsymbol{u}_j^{(n-1)} \overline{\times}_{n+1} \boldsymbol{u}_j^{(n+1)} \cdots \overline{\times}_N \boldsymbol{u}_j^{(N)} \\ &:= \underline{\boldsymbol{Y}}^{(j)} \,\overline{\times}_{-n} \{\boldsymbol{u}_j\}, \quad j = 1, \ldots, J; \; n = 1, \ldots, N. \end{aligned} \tag{50}$$

For simplicity, we use here a short notation $\underline{\boldsymbol{Y}}^{(j)} \,\overline{\times}_{-n} \{\boldsymbol{u}_j^T\}$ introduced by Kolda and Bader [28] to indicate multiplication of the tensor $\underline{\boldsymbol{Y}}$ by vectors in all modes, but n-mode. The above updating formula is elegant and relatively simple but involves rather high computational cost for large scale problems. In order to derive a more efficient (faster) algorithm we exploit basic properties the Khatri-Rao and Kronecker products of two vectors:

$$\left[\boldsymbol{U}^{(1)} \odot \boldsymbol{U}^{(2)}\right]_j = \left[\boldsymbol{u}_1^{(1)} \otimes \boldsymbol{u}_1^{(2)} \; \ldots \; \boldsymbol{u}_J^{(1)} \otimes \boldsymbol{u}_J^{(2)}\right]_j = \boldsymbol{u}_j^{(1)} \odot \boldsymbol{u}_j^{(2)}$$

or in more general form:

$$\left\{\boldsymbol{u}_j\right\}^{\odot_{-n}} = \left[\boldsymbol{U}^{\odot_{-n}}\right]_j. \tag{51}$$

Hence, by replacing $\boldsymbol{Y}_{(n)}^{(j)}$ terms in (49) by those in (45), and taking into account (51), the update learning rule (49) can be expressed as

$$\begin{aligned}\boldsymbol{u}_j^{(n)} &\leftarrow \boldsymbol{Y}_{(n)} \left[\boldsymbol{U}^{\odot_{-n}}\right]_j - \widehat{\boldsymbol{Y}}_{(n)} \left[\boldsymbol{U}^{\odot_{-n}}\right]_j + [\![\{\boldsymbol{u}_j\}]\!]_{(n)} \left\{\boldsymbol{u}_j\right\}^{\odot_{-n}} \\ &= [\boldsymbol{Y}_{(n)} \boldsymbol{U}^{\odot_{-n}}]_j - \boldsymbol{U}^{(n)} \boldsymbol{U}^{\odot_{-n}T} [\boldsymbol{U}^{\odot_{-n}}]_j + \boldsymbol{u}_j^{(n)} \left\{\boldsymbol{u}_j\right\}^{\odot_{-n}T} \left\{\boldsymbol{u}_j\right\}^{\odot_{-n}} \\ &= \left[\boldsymbol{Y}_{(n)} \boldsymbol{U}^{\odot_{-n}}\right]_j - \boldsymbol{U}^{(n)} \left[\boldsymbol{U}^{\odot_{-n}T} \boldsymbol{U}^{\odot_{-n}}\right]_j + \gamma_j^{(n)} \, \boldsymbol{u}_j^{(n)} \\ &= \left[\boldsymbol{Y}_{(n)} \boldsymbol{U}^{\odot_{-n}}\right]_j - \boldsymbol{U}^{(n)} \left[\left\{\boldsymbol{U}^T \boldsymbol{U}\right\}^{\circledast_{-n}}\right]_j + \gamma_j^{(n)} \, \boldsymbol{u}_j^{(n)} \\ &= [\boldsymbol{Y}_{(n)} \boldsymbol{U}^{\odot_{-n}}]_j - \boldsymbol{U}^{(n)} \left[\left\{\boldsymbol{U}^T \boldsymbol{U}\right\}^{\circledast} \oslash \left(\boldsymbol{U}^{(n)T} \boldsymbol{U}^{(n)}\right)\right]_j + \gamma_j^{(n)} \, \boldsymbol{u}_j^{(n)}, \end{aligned} \tag{52}$$

subject to the normalization of vectors $\boldsymbol{u}_j^{(n)}$ for $n = 1, 2, \ldots, N-1$ to unit length. In combination with a componentwise nonlinear half-wave rectifying operator, we finally have a new algorithm referred as the Fast HALS NTF algorithm:

$$\boldsymbol{u}_j^{(n)} \leftarrow \left[\gamma_j^{(n)} \, \boldsymbol{u}_j^{(n)} + [\boldsymbol{Y}_{(n)} \boldsymbol{U}^{\odot_{-n}}]_j - \boldsymbol{U}^{(n)} \left[\left\{\boldsymbol{U}^T \boldsymbol{U}\right\}^{\circledast} \oslash \left(\boldsymbol{U}^{(n)T} \boldsymbol{U}^{(n)}\right)\right]_j\right]_+. \tag{53}$$

The detailed pseudo-code of this algorithm is given in Algorithm 3. In a special case of $N = 2$, FAST-HALS NTF becomes FAST-HALS NMF algorithm described in the previous section.

Algorithm 3 FAST-HALS NTF

1: Nonnegative random or nonnegative ALS initialization $\boldsymbol{U}^{(n)}$ †
2: Normalize all $\boldsymbol{u}_j^{(n)}$ for $n = 1, \ldots, N-1$ to unit length
3: $\boldsymbol{T}_1 = (\boldsymbol{U}^{(1)T} \boldsymbol{U}^{(1)}) \circledast \ldots \circledast (\boldsymbol{U}^{(N)T} \boldsymbol{U}^{(N)})$
4: **repeat**
5: $\quad \boldsymbol{\gamma} = \text{diag}(\boldsymbol{U}^{(N)T} \boldsymbol{U}^{(N)})$
6: $\quad$ **for** $n = 1$ to N **do**
7: $\quad\quad \boldsymbol{\gamma} = \boldsymbol{1}$ if $n = N$
8: $\quad\quad \boldsymbol{T}_2 = \boldsymbol{Y}_{(n)} \{\boldsymbol{U}^{\odot_{-n}}\}$
9: $\quad\quad \boldsymbol{T}_3 = \boldsymbol{T}_1 \oslash (\boldsymbol{U}^{(n)T} \boldsymbol{U}^{(n)})$
10: $\quad\quad$ **for** $j = 1$ to J **do**
11: $\quad\quad\quad \boldsymbol{u}_j^{(n)} \Leftarrow \left[\gamma_j \, \boldsymbol{u}_j^{(n)} + [\boldsymbol{T}_2]_j - \boldsymbol{U}^{(n)} \; [\boldsymbol{T}_3]_j\right]_+$ ††
12: $\quad\quad\quad \boldsymbol{u}_j^{(n)} = \boldsymbol{u}_j^{(n)}/\|\boldsymbol{u}_j^{(n)}\|_2$ if $n \neq N$
13: $\quad\quad$ **end for**
14: $\quad\quad \boldsymbol{T}_1 = \boldsymbol{T}_3 \circledast \boldsymbol{U}^{(n)T} \boldsymbol{U}^{(n)}$
15: $\quad$ **end for**
16: **until** convergence criterion is reached

5. Flexible Local Algorithms Using Alpha Divergence

The algorithms derived in previous sections can be extended to more robust algorithms by applying a family of generalized Alpha and Beta divergences.

For the NMF problem (1) we define the Alpha divergence as follows (similar to [14], [18], [25], [29]):

$$D_\alpha^{(j)} \left(([\boldsymbol{Y}^{(j)}]_+) \parallel \boldsymbol{a}_j \underline{\boldsymbol{x}}_j\right) =$$

$$\begin{cases} \displaystyle\sum_{ik} \left(\frac{z_{ik}^{(j)}}{\alpha(\alpha+1)} \left[\left(\frac{z_{ik}^{(j)}}{y_{ik}^{(j)}} \right)^{\alpha} - 1 \right] - \frac{z_{ik}^{(j)} - y_{ik}^{(j)}}{\alpha+1} \right), & \alpha \neq -1, 0, \quad (54a) \\ \displaystyle\sum_{ik} \left((z_{ik}^{(j)}) \; \ln\left(\frac{z_{ik}^{(j)}}{y_{ik}^{(j)}} \right) - z_{ik}^{(j)} + y_{ik}^{(j)} \right), & \alpha = 0, \quad (54b) \\ \displaystyle\sum_{ik} \left(y_{ik}^{(j)} \ln\left(\frac{y_{ik}^{(j)}}{z_{ik}^{(j)}} \right) + z_{ik}^{(j)} - y_{ik}^{(j)} \right), & \alpha = -1, \quad (54c) \end{cases}$$

where $y_{ik}^{(j)} = [\boldsymbol{Y}]_{ik} - \sum_{p \neq j} a_{ip} x_{pk}$ and $z_{ik}^{(j)} = a_{ij} x_{jk} = a_{ij} b_{kj}$ for $j = 1, 2, \ldots, J$.

The choice of parameter $\alpha \in \mathbb{R}$ depends on statistical distributions of noise and data. In the special cases of the Alpha divergence for $\alpha = \{1, -0.5, -2\}$, we obtain respectively the Pearson's chi squared, Hellinger's, and Neyman's chi-square distances while for the cases $\alpha = 0$ and $\alpha = -1$, the divergence has to be defined by the limits of (54a) as $\alpha \to 0$ and $\alpha \to -1$, respectively. When these limits are evaluated for $\alpha \to 0$ we obtain the generalized Kullback-Leibler divergence defined by Eq. (54b) whereas for $\alpha \to -1$ we have the dual generalized Kullback-Leibler divergence given in Eq. (54c) [1], [14], [19], [25].

The gradient of the Alpha divergence (54) for $\alpha \neq -1$ with respect to a_{ij} and b_{kj} can be expressed in a compact form as:

$$\frac{\partial D_\alpha^{(j)}}{\partial b_{kj}} = \frac{1}{\alpha} \sum_i a_{ij} \left[\left(\frac{z_{ik}^{(j)}}{y_{ik}^{(j)}} \right)^{\alpha} - 1 \right], \tag{55}$$

$$\frac{\partial D_\alpha^{(j)}}{\partial a_{ij}} = \frac{1}{\alpha} \sum_k b_{kj} \left[\left(\frac{z_{ik}^{(j)}}{y_{it}^{(j)}} \right)^{\alpha} - 1 \right]. \tag{56}$$

† For 3-way tensor, direct trilinear decomposition could be used as initialization.

†† In practice, vectors $\boldsymbol{u}_j^{(n)}$ have often fixed sign before rectifying.

By equating the gradients to zero, we obtain a new multiplicative local α-HALS algorithm:

$$\boldsymbol{b}_j \leftarrow \left(\frac{\left[\boldsymbol{Y}^{(j)\,T}\right]_+^{.[\alpha]} \boldsymbol{a}_j}{\boldsymbol{a}_j^T \boldsymbol{a}_j^{.[\alpha]}} \right)^{.[1/\alpha]}, \quad \boldsymbol{a}_j \leftarrow \left(\frac{\left[\boldsymbol{Y}^{(j)}\right]_+^{.[\alpha]} \boldsymbol{b}_j}{\boldsymbol{b}_j^T \boldsymbol{b}_j^{.[\alpha]}} \right)^{.[1/\alpha]}, \quad (57)$$

where the "rise to the power" operations $\boldsymbol{x}^{.[\alpha]}$ are performed componentwise. The above algorithm can be generalized to the following form

$$\boldsymbol{b}_j \leftarrow \Psi^{-1}\left(\frac{\Psi\left(\left[\boldsymbol{Y}^{(j)T}\right]_+\right) \boldsymbol{a}_j}{\boldsymbol{a}_j^T\, \Psi(\boldsymbol{a}_j)} \right), \quad \boldsymbol{a}_j \leftarrow \Psi^{-1}\left(\frac{\Psi\left(\left[\boldsymbol{Y}^{(j)}\right]_+\right)\, \boldsymbol{b}_j}{\boldsymbol{b}_j^T \Psi(\boldsymbol{b}_j)} \right), \quad (58)$$

where $\Psi(\boldsymbol{x})$ is suitable chosen function, for example, $\Psi(\boldsymbol{x}) = \boldsymbol{x}^{.[\alpha]}$, componentwise[†].

In a similar way, novel learning rules for the N-order NTF problem (2) can be derived. For this purpose, we consider the n-mode matricized (unfolded) version of the tensor $\underline{\boldsymbol{Y}}$

$$\boldsymbol{Y}_{(n)} = \boldsymbol{U}^{(n)}\, (\boldsymbol{U}^{\odot_{-n}})^T. \quad (59)$$

Actually, this can be considered as an NMF model with $\boldsymbol{A} \equiv \boldsymbol{U}^{(n)}$ and $\boldsymbol{B} \equiv \boldsymbol{U}^{\odot_{-n}}$. From (51), we have

$$\boldsymbol{b}_j = \left[\boldsymbol{U}^{\odot_{-n}}\right]_j = \left\{\boldsymbol{u}_j\right\}^{\odot_{-n}}. \quad (60)$$

Applying directly the learning rule (58) to the model (59) gives

$$\boldsymbol{u}_j^{(n)} \leftarrow \Psi^{-1}\left(\frac{\Psi\left(\left[\boldsymbol{Y}_{(n)}^{(j)}\right]_+\right)\, \boldsymbol{b}_j}{\boldsymbol{b}_j^T \Psi(\boldsymbol{b}_j)} \right), \quad (61)$$

where $\boldsymbol{Y}_{(n)}^{(j)}$ is an n-mode matricized version of $\underline{\boldsymbol{Y}}^{(j)}$ in (45)

$$\begin{aligned} \boldsymbol{Y}_{(n)}^{(j)} &= \boldsymbol{Y}_{(n)} - \widehat{\boldsymbol{Y}}_{(n)} + \boldsymbol{u}_j^{(n)} \boldsymbol{b}_j^T = \boldsymbol{Y}_{(n)} - \widehat{\boldsymbol{Y}}_{(n)} + \boldsymbol{u}_j^{(n)} \left\{\boldsymbol{u}_j\right\}^{\odot_{-n}T} \\ &= \boldsymbol{Y}_{(n)} - \widehat{\boldsymbol{Y}}_{(n)} + [\![\{\boldsymbol{u}_j\}]\!]_{(n)}. \end{aligned} \quad (62)$$

For a specific nonlinear function $\Psi(\cdot)$ ($\Psi(x) = x^\alpha$)

$$\begin{aligned} \Psi(\boldsymbol{b}_j) &= \Psi(\{\boldsymbol{u}_j\}^{\odot_{-n}}) \\ &= \Psi(\boldsymbol{u}_j^{(N)}) \cdots \odot \Psi(\boldsymbol{u}_j^{(n+1)}) \odot \Psi(\boldsymbol{u}_j^{(n-1)}) \cdots \odot \Psi(\boldsymbol{u}_j^{(1)}) \\ &= \{\Psi(\boldsymbol{u}_j)\}^{\odot_{-n}}, \end{aligned} \quad (63)$$

and the denominator in (61) can be simplified as

$$\boldsymbol{b}_j^T\, \Psi(\boldsymbol{b}_j) = \{\boldsymbol{u}_j\}^{\odot_{-n}T} \{\Psi(\boldsymbol{u}_j)\}^{\odot_{-n}} = \{\boldsymbol{u}_j^T \Psi(\boldsymbol{u}_j)\}^{\circledast_{-n}}, \quad (64)$$

this completes the derivation of a flexible Alpha-HALS NTF update rule, which in the tensor form is given by

$$\boldsymbol{u}_j^{(n)} \leftarrow \Psi^{-1}\left[\frac{\Psi\left([\underline{\boldsymbol{Y}}^{(j)}]_+\right)\, \overline{\times}_{-n}\, \{\boldsymbol{u}_j\}}{\left\{\boldsymbol{u}_j^T \Psi(\boldsymbol{u}_j)\right\}^{\circledast_{-n}}} \right]_+, \quad (65)$$

where all nonlinear operations are componentwise[††] .

[†]For $\alpha = 0$ instead of $\Phi(x) = x^\alpha$ we used $\Phi(x) = \ln(x)$ [18].

[††]In practice, instead of half-wave rectifying we often use different transformations, e.g., real part of $\Psi(x)$ or adaptive nonnegative shrinkage function with gradually decreasing threshold till variance of noise σ^2_{noise}.

Algorithm 4 Alpha-HALS NTF

1: ALS or random initialization for all nonnegative vectors $\boldsymbol{u}_j^{(n)}$
2: Normalize all $\boldsymbol{u}_j^{(n)}$ for $n = 1, 2, ..., N-1$ to unit length,
3: Compute residue tensor $\underline{\boldsymbol{E}} = \underline{\boldsymbol{Y}} - [\![\{\boldsymbol{U}\}]\!] = \underline{\boldsymbol{Y}} - \widehat{\underline{\boldsymbol{Y}}}$
4: **repeat**
5: **for** $j = 1$ to J **do**
6: Compute $\underline{\boldsymbol{Y}}^{(j)} = \underline{\boldsymbol{E}} + \boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \ldots \circ \boldsymbol{u}_j^{(N)}$
7: **for** $n = 1$ to N **do**
8: $\boldsymbol{u}_j^{(n)}$ as in (65)
9: Normalize $\boldsymbol{u}_j^{(n)}$ to unit length vector if $n \neq N$
10: **end for**
11: Update $\underline{\boldsymbol{E}} = \underline{\boldsymbol{Y}}^{(j)} - \boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \ldots \circ \boldsymbol{u}_j^{(N)}$
12: **end for**
13: **until** convergence criterion is reached

6. Flexible HALS Algorithms Using Beta Divergence

Beta divergence can be considered as a flexible and complementary cost function to the Alpha divergence. In order to obtain local NMF algorithms we introduce the following definition of the Beta divergence (similar to [14], [18], [30]):

$$D_\beta^{(j)}([\boldsymbol{Y}^{(j)}]_+ \,\|\, \boldsymbol{a}_j \underline{\boldsymbol{x}}_j) =$$

$$\begin{cases} \displaystyle\sum_{ik} \left(([y_{ik}^{(j)}]_+)\, \frac{[y_{ik}^{(j)}]_+^\beta - z_{ik}^{(j)\beta}}{\beta} - \frac{[y_{ik}^{(j)}]_+^{\beta+1} - z_{ik}^{(j)\,\beta+1}}{\beta+1} \right), & \beta > 0, \quad (66a) \\ \displaystyle\sum_{ik} \left(([y_{ik}^{(j)}]_+)\, \ln\left(\frac{[y_{ik}^{(j)}]_+}{z_{ik}^{(j)}} \right) - [y_{ik}^{(j)}]_+ + z_{ik}^{(j)} \right), & \beta = 0, \quad (66b) \\ \displaystyle\sum_{ik} \left(\ln\left(\frac{z_{ik}^{(j)}}{[y_{ik}^{(j)}]_+} \right) + \frac{[y_{ik}^{(j)}]_+}{z_{ik}^{(j)}} - 1 \right), & \beta = -1, \quad (66c) \end{cases}$$

where $y_{ik}^{(j)} = y_{ik} - \sum_{p \neq j} a_{ip} b_{kp}$ and $z_{ik}^{(j)} = a_{ij} x_{jk} = a_{ij} b_{kj}$ for $j = 1, 2, \ldots, J$. The choice of the real-valued parameter $\beta \leq -1$ depends on the statistical distribution of data and the Beta divergence corresponds to Tweedie models [14], [19], [25], [30]. For example, if we consider the Maximum Likelihood (ML) approach (with no a priori assumptions) the optimal estimation consists of minimization of the Beta Divergence measure when noise is Gaussian with $\beta = 1$. For the Gamma distribution $\beta = -1$, for the Poisson distribution $\beta = 0$, and for the compound Poisson $\beta \in (-1, 0)$. However, the ML estimation is not optimal in the sense of a Bayesian approach where a priori information of sources and mixing matrix (sparsity, nonnegativity) can be imposed. It is interesting to note that the Beta divergence as special cases includes the standard squared Euclidean distance (for $\beta = 1$), the Itakura-Saito distance ($\beta = -1$), and the generalized Kullback-Leibler divergence ($\beta = 0$).

In order to derive a local learning algorithm, we compute the gradient of (66), with respect to elements to b_{kj}, a_{ij}:

Algorithm 5 Beta-HALS NTF

1: Initialize randomly all nonnegative factors $\boldsymbol{U}^{(n)}$
2: Normalize all $\boldsymbol{u}_{l,j}$ for $l = 1...N-1$ to unit length,
3: Compute residue tensor $\underline{\boldsymbol{E}} = \underline{\boldsymbol{Y}} - [\![\{\boldsymbol{U}\}]\!] = \underline{\boldsymbol{Y}} - \widehat{\underline{\boldsymbol{Y}}}$
4: **repeat**
5: **for** $j = 1$ to J **do**
6: Compute $\underline{\boldsymbol{Y}}^{(j)} = \underline{\boldsymbol{E}} + \boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \ldots \circ \boldsymbol{u}_j^{(N)}$
7: **for** $n = 1$ to $N-1$ **do**
8: $\boldsymbol{u}_j^{(n)} \Leftarrow \left[\underline{\boldsymbol{Y}}^{(j)} \,\overline{\times}_{-n}\, \{\Psi(\boldsymbol{u}_j)\}\right]_+$
9: Normalize $\boldsymbol{u}_j^{(n)}$ to unit length vector
10: **end for**
11: $\boldsymbol{u}_j^{(N)} \leftarrow \left[\dfrac{\underline{\boldsymbol{Y}}^{(j)} \,\overline{\times}_{-N}\, \{\Psi(\boldsymbol{u}_j)\}}{\{\Psi(\boldsymbol{u}_j)^T\, \boldsymbol{u}_j\}^{\circledast_{-n}}}\right]_+$
12: Update $\underline{\boldsymbol{E}} = \underline{\boldsymbol{Y}}^{(j)} - \boldsymbol{u}_j^{(1)} \circ \boldsymbol{u}_j^{(2)} \circ \ldots \circ \boldsymbol{u}_j^{(N)}$
13: **end for**
14: **until** convergence criterion is reached

$$\frac{\partial D_\beta^{(j)}}{\partial b_{kj}} = \sum_i \left(z_{ik}^{(j)\,\beta} - ([y_{ik}^{(j)}]_+)\, z_{ik}^{(j)\,\beta-1}\right) a_{ij}\,, \tag{67}$$

$$\frac{\partial D_\beta^{(j)}}{\partial a_{ij}} = \sum_k \left(z_{ik}^{(j)\,\beta} - ([y_{ik}^{(j)}]_+)\, z_{ik}^{(j)\,\beta-1}\right) b_{kj}. \tag{68}$$

By equating the gradient components to zero, we obtain a set of simple HALS updating rules referred to as the Beta-HALS algorithm:

$$b_{kj} \leftarrow \frac{1}{\sum_{i=1}^I a_{ij}^{\beta+1}} \sum_{i=1}^I a_{ij}^\beta\, ([y_{ik}^{(j)}]_+)\,, \tag{69}$$

$$a_{ij} \leftarrow \frac{1}{\sum_{k=1}^K b_{kj}^{\beta+1}} \sum_{k=1}^K b_{kj}^\beta\, ([y_{ik}^{(j)}]_+). \tag{70}$$

The above update rules can be written in a generalized compact vector form as

$$\boldsymbol{b}_j \leftarrow \frac{([\boldsymbol{Y}^{(j)\,T}]_+)\Psi(\boldsymbol{a}_j)}{\Psi(\boldsymbol{a}_j^T)\, \boldsymbol{a}_j}\,, \quad \boldsymbol{a}_j \leftarrow \frac{([\boldsymbol{Y}^{(j)}]_+)\, \Psi(\boldsymbol{b}_j)}{\Psi(\boldsymbol{b}_j^T)\, \boldsymbol{b}_j}\,, \tag{71}$$

where $\Psi(\boldsymbol{b})$ is a suitably chosen convex function (e.g., $\Psi(\boldsymbol{b}) = \boldsymbol{b}^{.[\beta]}$) and the nonlinear operations are performed element-wise.

The above learning rules could be generalized for the N-order NTF problem (2) (using the similar approach as for the Alpha-HALS NTF):

$$\boldsymbol{u}_j^{(n)} \leftarrow \frac{([\underline{\boldsymbol{Y}}_{(n)}^{(j)}]_+)\, \Psi(\boldsymbol{b}_j)}{\Psi(\boldsymbol{b}_j^T)\, \boldsymbol{b}_j}\,, \tag{72}$$

where $\boldsymbol{b}_j = \{\boldsymbol{u}_j\}^{\odot_{-n}}$, and $\boldsymbol{Y}_{(n)}^{(j)}$ are defined in (62) and (45).

By taking into account (63), the learning rule (72) can be written as follows

$$\boldsymbol{u}_j^{(n)} \leftarrow \frac{([\boldsymbol{Y}_{(n)}^{(j)}]_+)\, \{\Psi(\boldsymbol{u}_j)\}^{\odot_{-n}}}{\{\Psi(\boldsymbol{u}_j)\}^{\odot_{-n}\,T}\, \{\boldsymbol{u}_j\}^{\odot_{-n}}} = \frac{[\underline{\boldsymbol{Y}}^{(j)}]_+ \,\overline{\times}_{-n}\, \{\Psi(\boldsymbol{u}_j)\}}{\{\Psi(\boldsymbol{u}_j)^T\, \boldsymbol{u}_j\}^{\circledast_{-n}}} \tag{73}$$

Actually, the update rule (73) can be simplified to reduce computational cost by performing normalization of vectors $\boldsymbol{u}_j^{(n)}$ for $n = 1, \ldots, N-1$ to unit length vectors after each iteration step:

$$\boldsymbol{u}_j^{(n)} \leftarrow \left[\underline{\boldsymbol{Y}}^{(j)} \,\overline{\times}_{-n}\, \{\Psi(\boldsymbol{u}_j)\}\right]_+, \quad \boldsymbol{u}_j^{(n)} \leftarrow \boldsymbol{u}_j^{(n)} / \|\boldsymbol{u}_j^{(n)}\|_2. \tag{74}$$

The detailed pseudo-code of the Beta-HALS NTF algorithm is given in Algorithm 5. Once again, this algorithm can be rewritten in the fast form as follows

$$\boldsymbol{u}_j^{(n)} \leftarrow \left[\gamma_j^{(n)}\, \boldsymbol{u}_j^{(n)} + [\boldsymbol{Y}_{(n)}\, \{\Psi(\boldsymbol{U})\}^{\odot_{-n}}]_j - \boldsymbol{U}^{(n)} \left[\left\{\Psi(\boldsymbol{U})^T \boldsymbol{U}\right\}^{\circledast_{-n}}\right]_j\right]_+ \tag{75}$$

where $\gamma_j^{(n)} = \{\Psi(\boldsymbol{u}_j^T)\, \boldsymbol{u}_j\}^{\circledast_{-n}}, n = 1, \ldots, N$. The Fast HALS NTF algorithm is a special case with $\Psi(x) = x$.

In order to avoid local minima we have also developed a simple heuristic hierarchical Alpha- and Beta- HALS NTF algorithms combined with multi-start initializations using the ALS as follows:

1. Perform factorization of a tensor for any value of α or β parameters (preferably, set the value of the parameters to unity due to simplicity and high speed of the algorithm for this value).
2. If the algorithm has converged but has not achieved the desirable fit value (FIT max), restart the factorization by keeping the previously estimated factors as the initial matrices for the ALS initialization.
3. If the algorithm does not converge, alter the values of α or β parameters incrementally; this may help to overstep local minima.
4. Repeat the procedure until a desirable fit value is reached or there is a negligible or no change in the fit value or a negligible or no change in the factor matrices, or the value of the cost function in negligible or zero.

7. Simulation Results

Extensive simulations were performed for synthetic and real-world data on a 2.66 GHz Quad-Core Windows 64-bit PC with 8GB memory. For tensor factorization, the results were compared with some existing algorithms: the NMWF [31], the lsNTF [32] and also with two efficient implementations of general form of PARAFAC ALS algorithm by Kolda and Bader [16] (denoted as ALS_K) and by Andersson and Bro [33] (denoted as ALS_B). To make a fair comparison we apply the same stopping criteria and conditions: maximum difference of fit value, and we used three performance indexes: Peak Signal to Noise Ratio (PSNR) for all frontal slices, Signal to Interference Ratio (SIR)[†] for each columns of factors, and the explained variation ratio (i.e., how well the approximated tensor fit the input data tensor) for a whole tensor.

[†]The signal to interference ratio is defined as $SIR(\boldsymbol{a}_j, \hat{\boldsymbol{a}}_j) = 10\log(\|\boldsymbol{a}_j\|_2^2/(\|\boldsymbol{a}_j - \hat{\boldsymbol{a}}_j\|_2^2))$ for normalized and matched vectors.

7.1 Experiments for NMF

In **Example 1** we compare our HALS algorithms with the multiplicative Lee-Seung algorithm [34] and Chih-Lin Projected Gradient (PG) algorithm [35] for the benchmark `Xspectra` [36] (see Fig.3(b)). Ten mixtures were randomly generated from 5 sources (Fig.3(a)). We selected $\alpha = 1.5$ for α-HALS and $\beta = 2$ for β-HALS in order to show the difference in performance in comparison to the standard generalized Kullback-Leibler (K-L) divergence. Monte Carlo analysis was also performed with 100 trials and the average values of SIR for $\boldsymbol{X}$ and running time for each trial were summarized on Fig.3(c). Fast HALS NMF, α-HALS and β-HALS achieved higher performance than the two other well-known NMF algorithms. The simulation results for **Example 2** presented in Fig.4 were performed for the synthetic benchmark (Fig.4(a)) with 10 sparse (non-overlapping) nonnegative components. The sources were mixed by the randomly generated full column rank matrix $\boldsymbol{A} \in \mathbb{R}_{+}^{2\times 10}$, so only two mixed signals were available. The typical mixed signals are shown in Fig.4(b). The estimated components by the new β-HALS NMF algorithm (69)-(71) with $\beta = 0.1$ are illustrated in Fig.4(c). Moreover, the performance for different values of the parameter β are illustrated in Fig.4(d) and 4(e) with average Signal-to-Interference (SIR) level greater than 30 [dB]. Since the proposed algorithms (alternating technique) perform a non-convex optimization, the estimated components depend on the initial conditions. To estimate the performance in a statistical sense, we performed a Monte Carlo (MC) analysis. Figures 4(d) and 4(e) present the histograms of 100 mean-SIR samples for estimations matrices $\boldsymbol{A}$ and $\boldsymbol{X}$. We also conducted an experiment for the large scale similar problem in which we used 100 very sparse non-overlapped source signals and we mix them by random generated full column rank mixing matrix $\boldsymbol{A} \in \mathbb{R}_{+}^{2\times 100}$ (i.e., only two mixtures were used). Using the same algorithm but with 25 NMF layers, we were able to recover most of the sources in high probability. The performance is evaluated through the correlation matrix $\boldsymbol{R}_X = \hat{\boldsymbol{X}}\,\boldsymbol{X}^T$ which should be a diagonal matrix for a perfect estimation (given in Fig. 5(a)). Whereas distribution of the SIR performance is shown in Fig. 5(b). Detailed results are omitted due to space limits.

In **Example 3** we used five noisy mixtures of three smooth sources (benchmark signals `X_5smooth` [36]). Mixed signals were corrupted by additive Gaussian noise with SNR = 15 [dB] (Fig.6(a)). Fig.6 (c) illustrates efficiency of the HALS NMF algorithm with smoothness constraints using updates rules (41), including the Laplace operator $\boldsymbol{L}$ of the second order. The estimated components by the smooth HALS NMF using 3 layers [14] are depicted in Fig.6(b), whereas the results of the same algorithm with the smoothness constraint achieved $SIR\_A = 29.22$ [dB] and $SIR\_X = 15.53$ [dB] are shown in Fig.6(c).

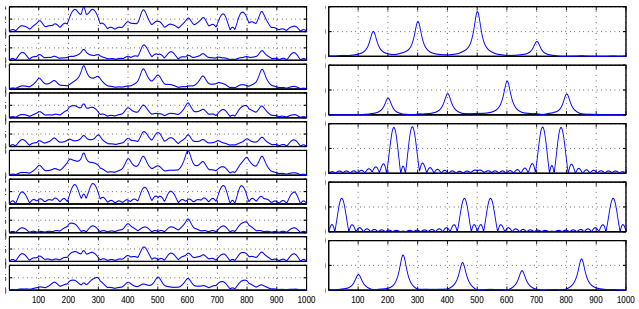
7.2 Experiments for NTF

In **Example 4**, we applied the NTF to a simple denoising of images. At first, a third-order tensor $\underline{\boldsymbol{Y}} \in \mathbb{R}_{+}^{51\times 51\times 40}$ whose each layer was generated by the L-shaped membrane function (which creates the MATLAB logo) $\underline{\boldsymbol{Y}}[:,:,k] = k*\texttt{membrane}(1,25), k = 1,\ldots,40$ has been corrupted by additive Gaussian noise with SNR 10 [dB] (Fig. 7(a)). Next, the noisy tensor data has been approximated by NTF model using our α-HALS and β-HALS algorithms with fit value 96.1%. Fig.7(a), 7(b) and 7(c) are surface visualizations of the 40-th noisy slice, and its reconstructed slices by $\alpha-$ and β-HALS NTF ($\alpha = 2, \beta = 2$), whereas Fig.7(d), 7(e) and 7(f) are their iso-surface visualizations, respectively. In addition, the performance for different values of parameters α and β are illustrated in Fig. 7(g) and 7(h) with PSNR in the left (blue) axis and number of iterations in the right (red) axis.

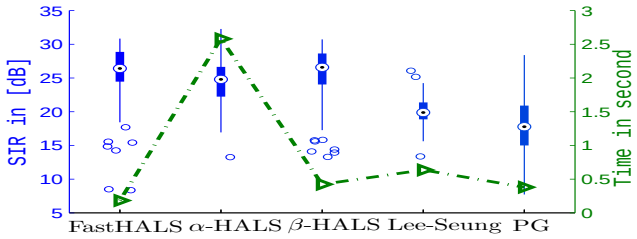
In **Example 5**, we constructed a large scale tensor with size of $500\times 500\times 500$ corrupted by additive Gaussian noise with SNR = 0 [dB] by using three benchmarks `X_spectra_sparse, ACPos24sparse10` and `X_spectra` [36] (see Fig.8(a)) and successfully reconstructed original sparse and smooth components using α- and β-HALS NTF algorithms. The performance is illustrated via volume, iso-surface and factor visualizations as shown in Fig. 8(b), 8(c) and 8(f); while running time and distributions of SIR and PSNR performance factors are depicted in Fig. 8(g). Slice 10 and its reconstructed slice are displayed in Fig.8(d) and 8(e). In comparison to the known NTF algorithms the Fast HALS NTF algorithm provides a higher accuracy for factor estimation based on SIR index, and the higher explained variation with the faster running time.

In **Example 6**, we tested the Fast HALS NTF algorithm for real-world data: Decomposition of amino acids fluorescence data (Fig.9(a)) from five samples containing tryptophan, phenylalanine, and tyrosine (`claus.mat`) [33], [37]. The data tensor was additionally corrupted by Gaussian noise with SNR = 0 dB (Fig.9(b)) , and the factors were estimated with $J = 3$. The β-HALS NTF was selected with $\beta = 1.2$, where for α-HALS NTF we select $\alpha = 0.9$. All algorithms were set to process the data with the same number of iterations (100 times). The performances and running times are compared in Fig. 10, and also in Table 3. In this example, we applied a smoothness constraint for Fast NTF, α- and β- HALS NTF. Based on fit ratio and PSNR index we see that, HALS algorithms usually exhibited better performance than standard NTF algorithms. For example, the first recovered slice (Fig.9(c)) is almost identical to the slice of the clean original tensor (99.51% Fit value). In comparison, the NMWF, lsNTF, ALS_K, ALS_B produced some artifacts as illustrated in Fig.9(d). Fig.9(e) and Fig.9(f).

In **Example 7** we used real EEG data: `tutorial-dataset2.zip` [38] which was pre-processed by complex Morlet wavelet. The tensor is represented by the inter-trial phase coherence (ITPC) for 14 subjects during a proprioceptive pull of left and right hand (28 files) with size 64

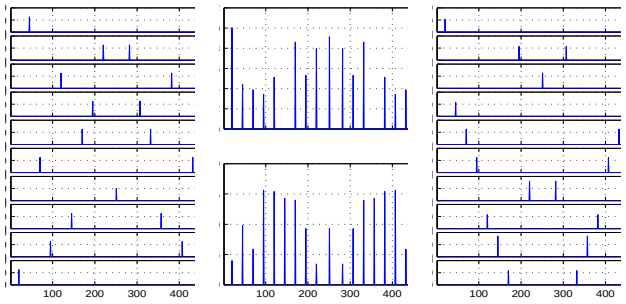


(a) 10 mixtures of dataset `Xspectra` (b) β-HALS ($\beta = 2$)

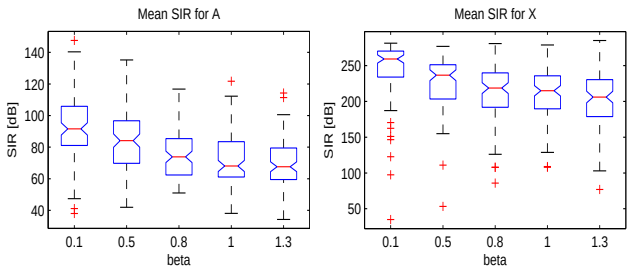


(c) SIR for $\boldsymbol{X}$ and running time

Fig. 3 Comparison of the Fast HALS NMF, α-HALS, β-HALS, Lee-Seung and PG algorithms in **Example 1** with the data set `Xspectra`. (a) observed mixed signals, (b) reconstructed original spectra (sources) using the β-HALS algorithm, (c) SIRs for the matrix $\boldsymbol{X}$ and computation time for different NMF algorithms.



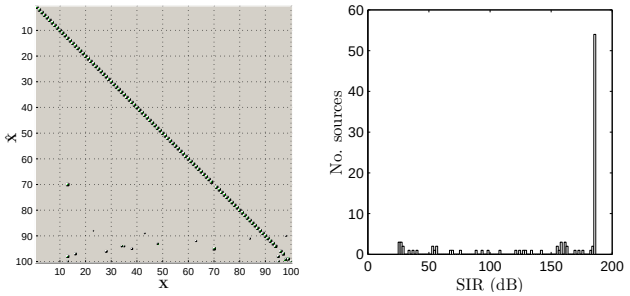
(a) 10 sources (b) 2 mixtures (c) β-HALS, $\beta = 0.1$



(d) SIR for $\boldsymbol{A}$ (e) SIR for $\boldsymbol{X}$

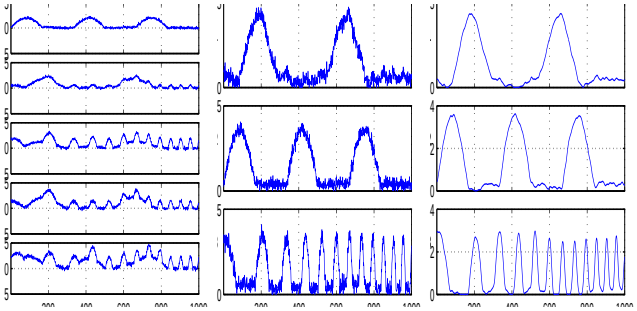
Fig. 4 Illustration of performance of the β-HALS NMF algorithm (a) 10 sparse sources assumed to be unknown, (b) two mixtures, (c) 10 estimated sources for $\beta = 0.1$. (d) & (e) SIR values for matrix $\boldsymbol{A}$ and sources $\boldsymbol{X}$ (respectively) obtained by the β-HALS NMF for $\beta = 0.1, 0.5, 0.8, 1, 1.3$ in the MC analysis of 100 trials.

$\times$ 4392 $\times$ 28. Exemplary results are shown in Fig.11 with scalp topographic maps and their corresponding IPTC time-frequency measurements and performance comparisons are



(a) correlation matrix $\boldsymbol{R} = \hat{\boldsymbol{X}}\boldsymbol{X}^T$ (b) SIR distribution

Fig. 5 Visualization of performance of extraction 100 sparse sources from only two linear mixtures for **Example 2**.



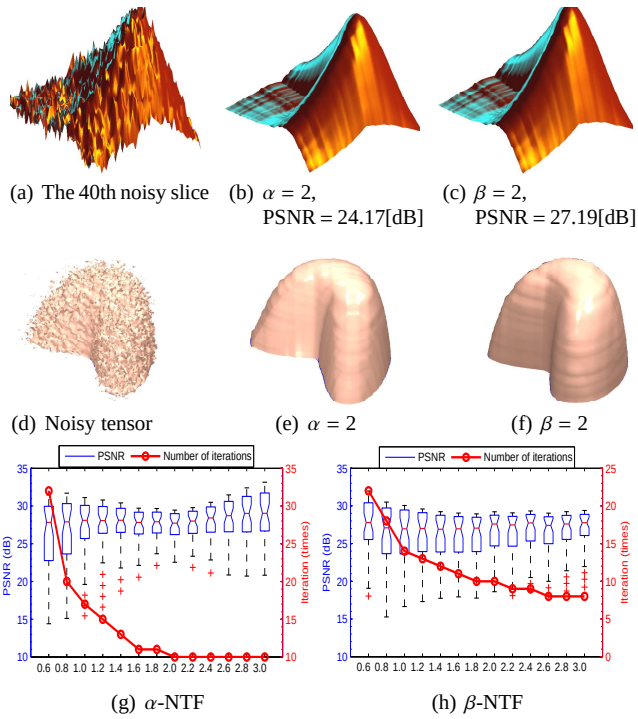
(a) Noisy mixtures, 10 dB Gaussian noise (b) Noisy estimated components (c) Smoothed components

Fig. 6 Illustration of performance of the regularized HALS NMF algorithm for **Example 3**.

given in Table 3. The components of the first factor $\boldsymbol{U}^{(1)}$ are relative to location of electrodes, and they are used to illustrate the scalp topographic maps (the first row in Fig.11); whereas the 2-nd factor $\boldsymbol{U}^{(2)}$ represents the frequency-time spectral maps which were vectorized, presented in the second row. Each component of these factors corresponds to a specific stimulus (left, right and both hands actions).

In **Example 8** we performed feature extraction for the CBCL face data set. The tensor was formed using the first 100 images of dimension 19 $\times$ 19 and then factorized by using 49 components and 100 components. The β-HALS NTF was selected with $\beta = 1$ to compare the HALS NTF algorithms with the NMWF and the lsNTF algorithm. For the case of 100 components, the reconstruction tensors explained 98.24 %, 97.83 % and 74.47% of the variation of the original tensor, for the β-HALS NTF, NMWF and lsNTF, respectively (Table 3). Note that the estimated components by using β-HALS NTF (Fig.12(b)) are relatively sparse and their reconstruction images are very similar to the original sources (Fig.12(a)).

Computer simulation for the above illustrated examples confirmed that the proposed algorithms give consistent and similar results to that obtained using the known "state of the arts" NMF/NTF algorithms, but our algorithms seem to be faster and more efficient. In other words, through extensive simulations we have confirmed that the FAST HALS NTF, α-HALS NTF and β-HALS NTF algorithms are robust to noise and produce generally better performance and provide faster convergence speed than existing recently de-

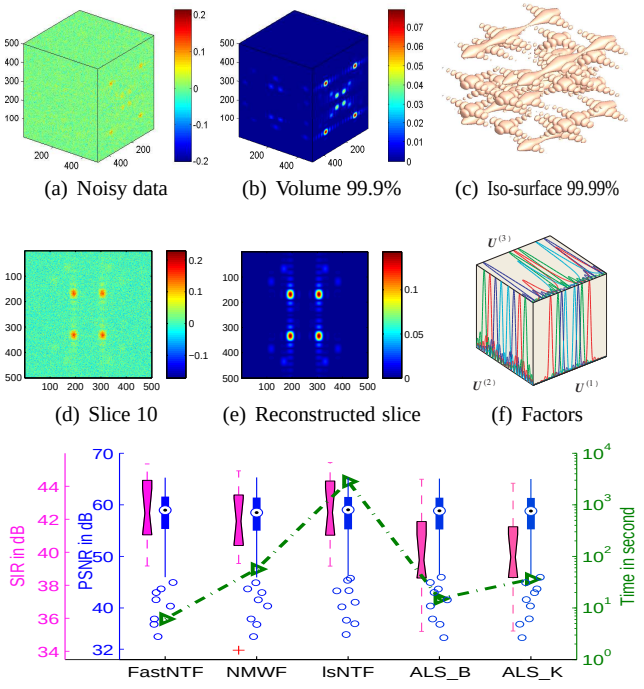


(a) The 40th noisy slice (b) $\alpha = 2$, PSNR = 24.17[dB] (c) $\beta = 2$, PSNR = 27.19[dB]

(d) Noisy tensor (e) $\alpha = 2$ (f) $\beta = 2$

(g) α-NTF (h) β-NTF

Fig. 7 Illustration of data reconstruction for noisy tensor $\underline{\boldsymbol{Y}} \in \mathbb{R}_{+}^{51\times51\times40}$ for **Example 4**: (a), (b) & (c) surface visualizations of the 40th noisy slice and its reconstructed slices by α- and β-HALS NTF algorithms ($\alpha = 2$, $\beta = 2$), respectively; (d)-(f) iso-surface visualizations of noisy tensor and its reconstructed tensors by α- and β-HALS-NTF algorithms; (g) & (h) Performance of the HALS NTF algorithms for different values of α and β but for the same desired fit value 96.1%.

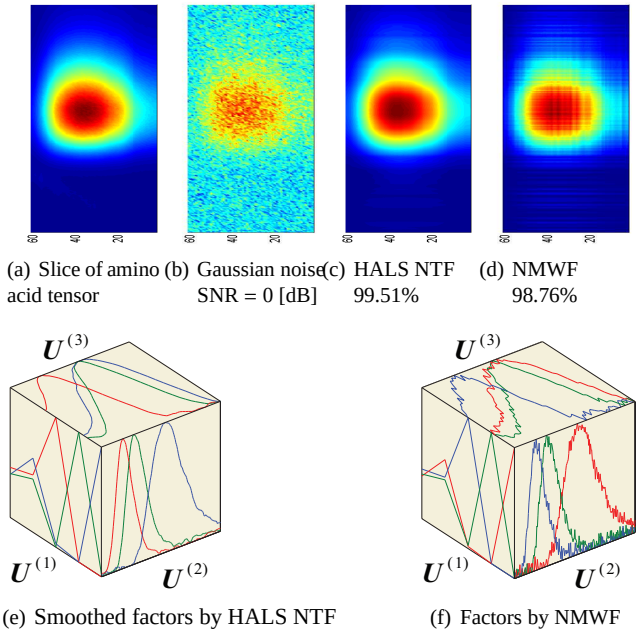


(a) Noisy data (b) Volume 99.9% (c) Iso-surface 99.99%

(d) Slice 10 (e) Reconstructed slice (f) Factors

(g) Performance comparison

Fig. 8 Illustration of tensor reconstruction by Fast HALS NTF for **Example 5** with tensor $\underline{\boldsymbol{Y}} \in \mathbb{R}_{+}^{500\times500\times500}$ degraded by Gaussian noise with SNR = 0[dB].



(a) Slice of amino acid tensor (b) Gaussian noise SNR = 0 [dB] (c) HALS NTF 99.51% (d) NMWF 98.76%

(e) Smoothed factors by HALS NTF (f) Factors by NMWF

Fig. 9 Illustration of estimated factors by the FAST-HALS NTF in comparison to the NMWF algorithm for three-way decomposition of amino acid data in Example 6. (a) The first slice of original tensor, (b) The same slice with huge Gaussian noise, (c)-(d) the reconstructed slices using HALS NTF and NMWF, (e)-(f) three estimated factors using HALS and NMWF algorithms (The estimated factors should be as smooth as possible).

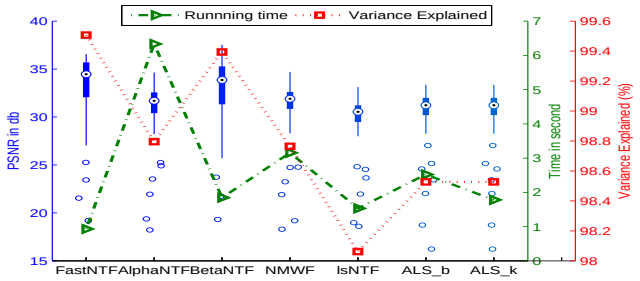
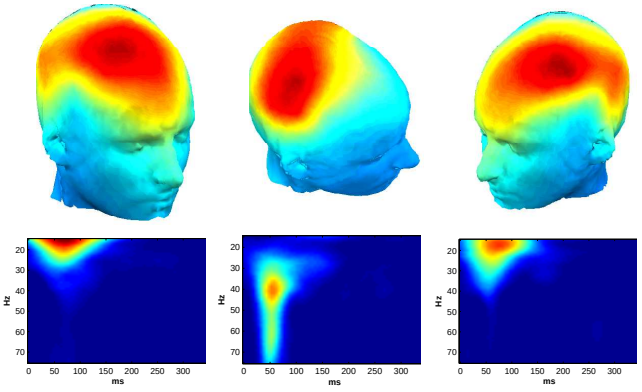
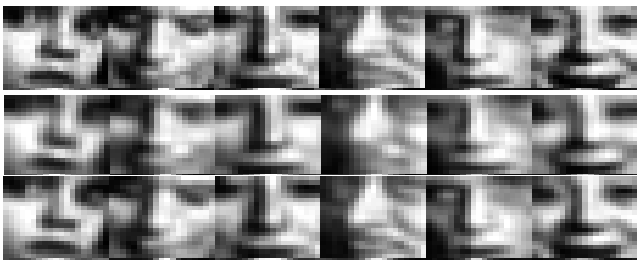


Fig. 10 Comparison of performance and running time for amino acid data with tensor $\underline{\boldsymbol{Y}} \in \mathbb{R}_{+}^{5\times201\times61}$ corrupted by Gaussian noise with SNR = 0[dB].

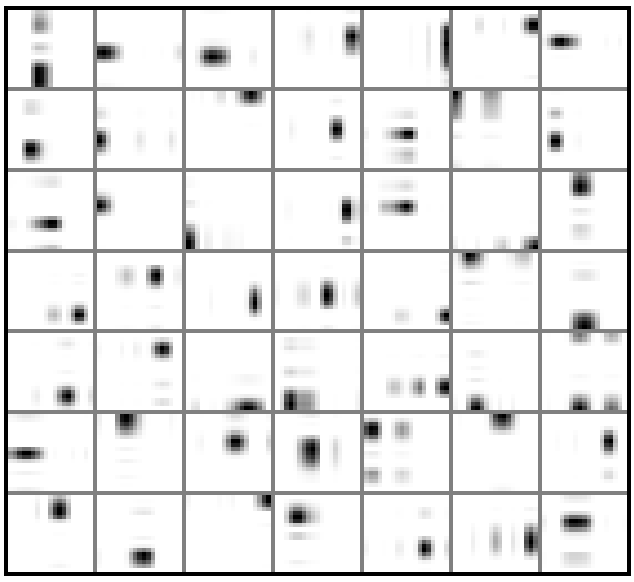


(a) Left hand stimuli (b) Gamma activity of both stimuli (c) Right hand stimuli

Fig. 11 EEG analysis using the FAST HALS NTF for **Example 7** with factor matrices for $\boldsymbol{U}^{(1)}$ for a scalp topographic map (first row), factor $\boldsymbol{U}^{(2)}$ for spectral (time-frequency) map (second row) (see [38] for details). Results are consistent with previous analysis [38] but run time is almost 8 times shorter and fit is slightly better.



(a) 6 original CBCL images (top) and their reconstructions by 49 components (94.81%) (center) and 100 components (98.24%) (bottom).



(b) 49 basis components estimated by β-HALS NTF, 94.95 % (Fit).

Fig. 12 Illustration of factorization of 100 CBCL face images into 49 and 100 basis components by using the β-HALS NTF algorithm.

veloped NMF/NTF algorithms.

8. Conclusions and Discussion

The main objective and motivations of this paper is to derive fast and efficient algorithms for NMF/NTF problems. The extended algorithms are verified for many different benchmarks. The developed algorithms are robust to noisy data and have many potential applications. These algorithms are also suitable to large scale dataset due to their local learning rules, and fast processing speed. The algorithms can be extended to semi-NTF and to sparse PARAFAC using suitable nonlinear projections and regularization terms [17]. These are the unique extensions of the standard NMF HALS algorithm, and to the authors' best knowledge, the first time such algorithms have been applied and practically implemented to multi-way NTF models. We have implemented the proposed algorithms in MATLAB in our toolboxes NMFLAB/NTFLAB and they will be available soon free for researchers [5]. The performance of the developed algorithms are compared with some of the existing NMF and NTF algorithms. The proposed algorithms are shown to be superior in terms of performance, speed and convergence properties. Of

Table 2 Description of data sets and notation of Examples

No.	Data set	Size	J
4	L-shaped membrane function, MATLAB logo	$51 \times 51 \times 40$	4
5	`X_spectra_sparse`, `ACPos24sparse10` and `X_spectra` [36]	$500 \times 500 \times 500$	4
6	Amino acids fluorescence data, `claus.mat` [37]	$5 \times 201 \times 61$	5
7	ITPC of 14 subjects during a proprioceptive pull of left and right hand (28 datasets), $64 channels \times (61 frequency - 72 time) \times 28 subjects$, `tutorialdataset2.set`[38]	$64 \times 4392 \times 28$	3
8	`MIT CBCL` face images	$190 \times 19 \times 100$	49 100

Table 3 Comparison of Performance of NTF Algorithms for Examples 5-9

	Fit (%)				**Time** (seconds)		
Example No.	5	6	7	8	5	6	7
FastNTF	**99.9955**	**99.51**	**52.41**		**51.73**	**0.93**	**7.08**
α-NTF		98.77			6.33		
β-NTF	99.9947	99.39		**98.24**	470.53	1.85	
NMWF †	99.9918	98.76	52.38	97.83	513.37	3.16	58.19
lsNTF ††	✗	98.06	51.33	74.47	✗	3.30	4029.84
ALS_B	99.9953	98.53	**53.17**		145.73	2.52	67.24
ALS_K	99.9953	98.53	53.13		965.76	1.78	66.39

course, there are still many open theoretical problems like global convergence of the algorithms and optimal choice of α and β parameters.

Acknowledgment

The authors would like to thank the associate editor Professor Kazushi Ikeda and anonymous reviewers for their valuable comments and helpful suggestions that greatly improves this paper's quality.

References

[1] S. Amari, Differential-Geometrical Methods in Statistics, Springer Verlag, 1985.

[2] D.D. Lee and H.S. Seung, "Learning the parts of objects by non-negative matrix factorization," Nature, vol.401, pp.788–791, 1999.

[3] A. Cichocki, R. Zdunek, and S.I. Amari, "Hierarchical ALS algorithms for nonnegative matrix and 3D tensor factorization," Springer LNCS, vol.4666, pp.169–176, 2007.

[4] A. Cichocki, R. Zdunek, and S. Amari, "Csiszar's divergences for non-negative matrix factorization: Family of new algorithms," Springer LNCS, vol.3889, pp.32–39, 2006.

[5] A. Cichocki, R. Zdunek, A.H. Phan, and S. Amari, Nonnegative Matrix and Tensor Facorizarions and Beyond, Wiely, Chichester, 2009.

[6] M. Mørup, L.K. Hansen, C.S. Herrmann, J. Parnas, and S.M. Arnfred, "Parallel factor analysis as an exploratory tool for wavelet transformed event-related EEG," NeuroImage, vol.29, no.3, pp.938–947, 2006.

[7] F. Miwakeichi, E. Martnez-Montes, P. Valds-Sosa, N. Nishiyama, H. Mizuhara, and Y. Yamaguchi, "Decomposing EEG data into

†In fact, the NMWF failed for very noisy data due to large negative entries. We enforced the estimated components to have nonnegative values by half-wave rectifying.

††lsNTF failed for large scale example with tensor of $500 \times 500 \times 500$. However, for the same problem with a reduced dimension of tensor: $300 \times 300 \times 300$, lsNTF needed 2829.9800 seconds and achieved 99.9866% of fit value, so our algorithm was at least 50 times faster.

space−time−frequency components using parallel factor analysis," NeuroImage, vol.22, no.3, pp.1035–1045, 2004.

[8] A. Shashua, R. Zass, and T. Hazan, "Multi-way clustering using super-symmetric non-negative tensor factorization," European Conference on Computer Vision (ECCV), Graz, Austria, May 2006.

[9] J. Sun, D. Tao, and C. Faloutsos, "Beyond streams and graphs: dynamic tensor analysis," Proc.of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp.374–383, 2006.

[10] M. Heiler and C. Schnoerr, "Controlling sparseness in non-negative tensor factorization," Springer LNCS, vol.3951, pp.56–67, 2006.

[11] T. Hazan, S. Polak, and A. Shashua, "Sparse image coding using a 3D non-negative tensor factorization," International Conference of Computer Vision (ICCV), pp.50–57, 2005.

[12] A. Smilde, R. Bro, and P. Geladi, Multi-way Analysis: Applications in the Chemical Sciences, John Wiley and Sons, New York, 2004.

[13] M. Berry, M. Browne, A. Langville, P. Pauca, and R. Plemmons, "Algorithms and applications for approximate nonnegative matrix factorization," Computational Statistics and Data Analysis, vol.52, no.1, pp.155–173, 2007.

[14] A. Cichocki, R. Zdunek, S. Choi, R. Plemmons, and S. Amari, "Nonnegative tensor factorization using Alpha and Beta divergencies," Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP07), Honolulu, Hawaii, USA, pp.1393–1396, April 15–20 2007.

[15] P. Sajda, S. Du, T. Brown, L. Parra, and R. Stoyanova, "Recovery of constituent spectra in 3d chemical shift imaging using nonnegative matrix factorization," 4th International Symposium on Independent Component Analysis and Blind Signal Separation, Nara, Japan, pp.71–76, April 2003.

[16] T.G. Kolda and B. Bader, "Tensor decompositions and applications," SIAM Review, June 2008.

[17] A. Cichocki, A.H. Phan, R. Zdunek, and L.Q. Zhang, "Flexible component analysis for sparse, smooth, nonnegative coding or representation," Lecture Notes in Computer Science, pp.811–820, Springer, 2008.

[18] A. Cichocki, S. Amari, R. Zdunek, R. Kompass, G. Hori, and Z. He, "Extended SMART algorithms for non-negative matrix factorization," Springer LNAI, vol.4029, pp.548–562, 2006.

[19] A. Cichocki and R. Zdunek, "Regularized alternating least squares algorithms for non-negative matrix/tensor factorizations," Springer LNCS, vol.4493, pp.793–802, June 3–7 2007.

[20] N.D. Ho, Nonnegative Matrix Factorization - Algorithms and Applications, thse/dissertation, FSA/INMA - Dpartement d'ingnierie mathmatique, 2008.

[21] N.D. Ho, P.V. Dooren, and V. Blondel, "Descent algorithms for nonnegative matrix factorization," Numerical Linear Algebra in Signals, Systems and Control, 2008. to appear.

[22] M. Biggs, A. Ghodsi, and S. Vavasis, "Nonnegative matrix factorization via rank-one downdate," ICML-2008, Helsinki, July 2008.

[23] N. Gillis and F. Glineur, "Nonnegative matrix factorization and underapproximation," SIAM conference on Optimization, Boston, May 2008. Preprint.

[24] A.H. Phan and A. Cichocki, "Multi-way Nonnegative Tensor Factorization Using Fast Hierarchical Alternating Least Squares Algorithm (HALS)," Proc. of The 2008 International Symposium on Nonlinear Theory and its Applications, Budapest, Hungary, 2008.

[25] A. Cichocki, R. Zdunek, S. Choi, R. Plemmons, and S.I. Amari, "Novel multi-layer nonnegative tensor factorization with sparsity constraints," Springer LNCS, vol.4432, pp.271–280, April 11–14 2007.

[26] M. Nikolova, "Minimizers of cost-functions involving nonsmooth data-fidelity terms. application to the processing of outliers," SIAM J. Numer. Anal., vol.40, no.3, pp.965–994, 2002.

[27] P.C. Hansen, "Regularization tools version 3.0 for matlab 5.2," Numerical Algorithms, vol.20, pp.195–196, 1999.

[28] B.W. Bader and T.G. Kolda, "Algorithm 862: Matlab tensor classes for fast algorithm prototyping," ACM Trans. Math. Softw., vol.32, no.4, pp.635–653, 2006.

[29] A. Cichocki, A.H. Phan, and C. Caiafa, "Flexible HALS algorithms for sparse non-negative matrix/tensor factorization," Proc. of The eighteenth of a series of IEEE workshops on Machine Learning for Signal Processing, Cancun, Mexico, 16–19, October 2008.

[30] M. Minami and S. Eguchi, "Robust blind source separation by Beta-divergence," Neural Computation, vol.14, pp.1859–1886, 2002.

[31] M. Mørup, L.K. Hansen, J. Parnas, and S.M. Arnfred, "Decomposing the time-frequency representation of EEG using non-negative matrix and multi-way factorization," tech. rep., 2006.

[32] M.P. Friedlander and K. Hatz, "Computing nonnegative tensor factorizations," Tech. Rep. TR-200621, Dept. Computer Science, University of British Columbia, Vancouver, December 2007. To appear in *Optimization Methods and Software*.

[33] C.A. Andersson and R. Bro, "The *N*-way Toolbox for MATLAB," Chemometrics Intell. Lab. Systems, vol.52, pp.1–4, 2000.

[34] D.D. Lee and H.S. Seung, Algorithms for nonnegative matrix factorization, NIPS, MIT Press, 2001.

[35] C.J. Lin, "Projected gradient methods for non-negative matrix factorization," Neural Computation, vol.19, no.10, pp.2756–2779, October 2007.

[36] A. Cichocki and R. Zdunek, "NMFLAB for Signal and Image Processing," tech. rep., Laboratory for Advanced Brain Signal Processing, BSI, RIKEN, Saitama, Japan, 2006.

[37] R. Bro, "PARAFAC. Tutorial and applications," Special Issue 2nd Internet Conf. in Chemometrics (INCINC'96), pp.149–171, Chemom. Intell. Lab. Syst, 1997.

[38] M. Mørup, L.K. Hansen, and S.M. Arnfred, "ERPWAVELAB a toolbox for multi-channel analysis of time-frequency transformed event related potentials," Journal of Neuroscience Methods, vol.161, pp.361–368, 2007.

Andrzej Cichocki was born in Poland. He received his M.Sc. (with honors), Ph.D. and Habilitate Doctorate (Dr.Sc.) degrees, all in electrical engineering, from the Warsaw University of Technology (Poland). He is the co-author of four international books and monographs (two of them translated to Chinese): Nonnegative Matrix and Tensor Factorizations and Beyond, J. Wiley 2009, Adaptive Blind Signal and Image Processing, J. Wiley 2002, MOS Switched-Capacitor and Continuous-Time Integrated Circuits and Systems (Springer-Verlag, 1989) and Neural Networks for Optimization and Signal Processing (J. Wiley and Teubner Verlag, 1993/94) and author or co-author of more than two hundreds papers. He is Editor-in-Chief of Journal Computational Intelligence and Neuroscience. Currently, he is the head of the laboratory for Advanced Brain Signal Processing in the RIKEN Brain Science Institute, Japan.

Anh Huy Phan received his B.E. and M.Sc. degrees from the Ho-Chi-Minh City University of Technologies, in area of Electronic Engineering, He worked as Deputy Head of Research and Development Department, Broadcast Research and Application Center, Vietnam Television; also, taught as lecturer part-time at Van Hien University, Hong Bang University, Electronic and Computer Center of University of Natural Sciences in Ho-Chi-Minh City, Vietnam, in areas of Probability and Statistics, Numerical Algorithms, MATLAB Programming. Actually he is working as the technical staff in the Laboratory for Advanced Brain Signal Processing and he is doing research towards his Ph.D. degree under supervision of Professor Cichocki.